STUDY OF PATTERN RECOGNITION TECHNIQUES FOR NOISY ENGLISH ALPHABETS & WORDS WITH SOFT COMPUTING TECHNIQUES

SUBMITTED TOWARDS THE FULFILLMENT OF
DEGREE
OF
DOCTOR OF PHILOSOPHY
IN
MATH SCI & COMPUTER APPLICATIONS

BY

SAURABH SHRIVASTAVA

BUNDELKHAND UNIVERSITY
Central Library
ACC. No t 3025
Date 13-09-10
JHANSI

UNDER THE SUPERVISION OF

GUIDE:

Prof. P. N. SRIVASTAVA
Dean Academics
College of Science & Engg.
JHANSI (UP)

CO – GUIDE:

Dr M. P. SINGH
Sr. Lecturer
ICIS, Khandari Campus,
Dr. B.R.Ambedkar University, Agra (U.P.)

2008

Department of Mathematical Sciences & Computer Applications
Bundelkhand University
Jhansi 284128

Certificate

This is to certify that ***Mr. Saurabh Shrivastava***, has completed his doctoral dissertation on the topic entitled" ***Study of Pattern Recognition Techniques for Noisy English Alphabets and Words with Soft Computing Techniques***", under our supervision and guidance in the Department of Mathematical Sciences and Computer Applications, Bundelkhand University, Jhansi. He was completed his work under my presence 200 days in the same department.

To best of our knowledge it is an original research work made by the candidate. The thesis is worthy of consideration for the award of Degree of Doctor of Philosophy in Computer Application.

No part of this thesis has been submitted for any degree or diploma.

[illegible]

Dr Manu Pratap Singh
Sr. Lecturer
Institute of Computer & Information Science, Khandari Campus ,
Dr B R Ambedkar University, Agra(UP).

[illegible]

Prof P N Srivastava
Professor & Dean
College of Science & Engineering,
Ambabay, Jhansi(UP) .

DECLARATION

I, Saurabh Shrivastava, the research scholar hereby declare that this research entitled***, "Study of Pattern Recognition Techniques for Noisy English Alphabets & Words with Soft Computing Techniques"***, is an original research work of mine and has not been submitted earlier for award of any degree or diploma.

[illegible]

April 21, 2008 **(Saurabh Shrivastava)**

Acknowledgement

Gurur Brahmma Gurur Vishnu Gurur Devo Maheshwaray
Guru Sakchhat Parambrhamma Tasmai Shri Guruveh Namah

These two lines sum up my thoughts when I write the acknowledgement for my thesis. No work is possible without the blessing and the support of the guide under whom one has started the research work; My heartfelt thanks and gratitude first of all go to my Guide, Prof.P.N.Srivastava and Dr.M.P.Singh. The constant support, encouragement, and guidance provided by them has finally culminated today in the form of this thesis. I shall be forever indebted to both of them. I also like to thank all teachers of mine who taught me since starting of my education.

It is also the unfailing support and belief of my mother and father, who have helped me to achieve this milestone in my life. I thank them from the core of my heart. Their painstaking efforts have constantly motivated me.

I would like to thank Sri. V. K. Mittal, Vice-Chancellor, Bundelkhand University, Jhansi for his constant support and encouragement in working on new projects. Sincere thanks are due to Prof. R. P. Agarwal for his help and encouragement in getting the work done. My thanks are also due to Prof. V. K. Sehgal, Head, Department of Mathematical Sciences and Computer Applications, for his constant support, positive criticism and guidance in the completion of the research work. The moral support and encouragement of Dr Pankaj Atri was a great support in starting the research work. The constant encouragement of Dr Aparna Raj was a great help in the completion of this research work. The research support of Dr Ajit and Dr Manish Mangal was too precious for the completion of this work; I would like to thank them. Friends were always with me to support me for the completion of this work, I like to express my sincere thanks for all of them, specially of Dr. D. K. Bhatt, Deepak Tomar, Dr. Yashodhara Sharma,Dr Alok Verma, Ashish Srivastava, Prabhakar Khare, Dr. Avneesh Kumar, Dr. mamta Singh, Dr. A.p. Singh, Dr. Rishi Saxena, Khusendra Borkar, Sharad Gaur, Shishir Saxena, Rupesh Gupta, Nikhil saxena, Keshav Kansana, Shashank Khare.

I am very much thankfull to Smt. Arti Srivastava, Dr. Saket , Nupur ,Dr. Kislay, Kiran, for their encouragement and pursuance to complete my research work.

Support of family is always required to begin and conclude any work. I am very much thankful to my wife Ruchi, for her never-ending love, support and patient bearing, my brothers Shishir and Sharad, to give me energy, my bhabhis Reena and Punita, for giving a pleasant atmosphere, my sister Dr. Ritu and her husband Vishal, for their encouragement, my father in law Sri. R.K..Srivastava for his support, my brother and sister in laws Amit, Ritu and Richa, for their love and affections and all the kids of my family Varid,, Aarush, Agam, Atharva and Vaibhav, for providing me the joyful moments so that I could relax my self.

I would like to thank all those who have helped me directly or indirectly in the completion of this work. The facilities provided by the University are duly acknowledged.

(Saurabh Shrivastava)

April, 2008

Contents

CHAPTER 1

Preliminary Mathematical Concepts

Chapter 1: Preliminary Mathematical Concepts

Mathematics provides the powerful tools for establishing any theory and idea. The mathematical modeling helps to define the neural networks and their functionalities. The basic operations of the neurons and its topological structures can easily specify with the help of mathematical modeling. The input/ output patterns for the neural network can represent with vector and the various learning methods of neural networks are define to employ the mathematical tools like differential equation, gradient methods, matrix operations and many more. In this chapter we are discussing [1-7] the basic preliminaries of mathematical tools, those are used very frequently to understand the neural network architecture and its learning techniques accomplish the task of pattern recognition.

1.1 Sets

A set is a collection of objects or elements, sharing the common characteristics or properties. A set is denoted by listing the elements between braces. The set of positive integers is $\{1,2,3,\ldots\}$. We also denote a sets with the notation {x | conditions on x} for sets that are more easily described than enumerated. This is read as "the set of elements x such that x satisfies..." $x \in S$ is the notation for "x is an element of the set S". To express the opposite we have $x \notin S$ for "x is not an element of the set S".

The following is the summary to define the sets adequately:

- The ***empty set*** is defined as a set with nothing. It may be denoted as $\{\ \}$ or ϕ.
- Two sets A and B are ***equal*** if they contain exactly the same elements.
- Two sets A and B are said be ***disjoint***, if they have no elements in common.

- A is ***subset*** of B if every element in A also in B, it is denoted as $A \subseteq B$.
- The cardinal number or cardinality of a finite set is non negative integer representing the number of elements in the set. This is denoted as $|A|$.
- The ***complement*** of a set A is defined as a set of all elements (of a special set S, the space or universe) that are not in A and it is denoted as $\overline{A}$.
- ***Intersection*** of set A and B is defined as:

$$A \cap B = \left\{ x \middle| x \in A \quad and \quad x \in B \right\}.$$

- ***Union*** of two sets A and B is defined as

$$A \cup B = \left\{ x \middle| x \in A \quad or \quad x \in B \right\}$$

- ***Differnce*** of two sets A and B is defined as

$$A - B = \{x | x \in A \quad and \quad x \notin B\}$$

- A partition P of a set A is a collection of mutually exclusive subsets of A_i that

satisfy : $A_i \cap A_j \quad unless \quad i = j$ and $\bigcup_i A_i = A$

1.2 Relations

Relations are based on the notion of set mapping and provide a mathematical formalism for the representation of structure.

If A and B are sets, a relation from A to B is a subset of $A \times B$. Where $A \times B$ denotes the Cartesian product of the set A and B. Given a set

$$A = \{a, b, c, ...\} \quad \text{and} \quad B = \{x, y, z, ...\},$$

a relation from A to B, namely *R*, satisfies $R \subseteq A \times B$.

This is defined as a binary relation, since it involves only two sets and provides a way of connecting or relating members of the sets. The relations can also be defined as sets of ordered pair say (x, y), here set of all possible values of x is the ***domain*** and the set of all possible values of y is called the ***range***. The relations have a direction or ordering, this directionality and ordering of a relation defines the following important properties:

- ***Reflexive:*** *R* is reflexive if, $\forall a \in A, (a, a) \in R.$
- ***Symmetric:*** *R* is symmetric if, $\forall (a, b) \in R, (b, a) \in R.$
- ***Transitive:*** *R* is transitive if, $\forall (a, b) \in R \quad and \quad (b, c) \in R \quad then \quad (a, c) \in R$
- ***Equivalence Relations:*** A ***relation*** that satisfies all three above mentioned properties, is termed as an equivalence relation.

1.3 Functions

A function f from A to B is a relation such that for every $a \in A, \exists$ one and only one $b \in B, such\ that\ (a, b) \in f$. Usually it has been shown as:

$$f : A \Rightarrow B,$$

where A is the domain of function *f* and B is the range of the function. If $(a,b)\in f$, we say that *b* is the function value of *a,* and denoted as:

$$b=f(a)$$

If $b=f(a)$, then we can write $a=f^{-1}(b)$ where f^{-1} is the ***inverse*** of f. If $b=f(a)$ is a one-to-one function, then $f^{-1}(b)$ is also one-to-one function. In this case, $a=f^{-1}(f(a))=f(f^{-1}(x))$ for each value of *a* where both $f(a)$ and $f^{-1}(a)$ are defined. If $b=f(a)$ is a many-to-one function, then $a=f^{-1}(b)$ is a one-to-many function. $f^{-1}(b)$ is a multi-valued function.

1.4 Scalars and Vectors

A vector is a quantity having both magnitude and a direction. Examples of vector quantities are velocity, force and position. One can represent a vector in n-dimensional space with an arrow whose initial point is at the origin (figure 1.1). The magnitude is the length of the vector.

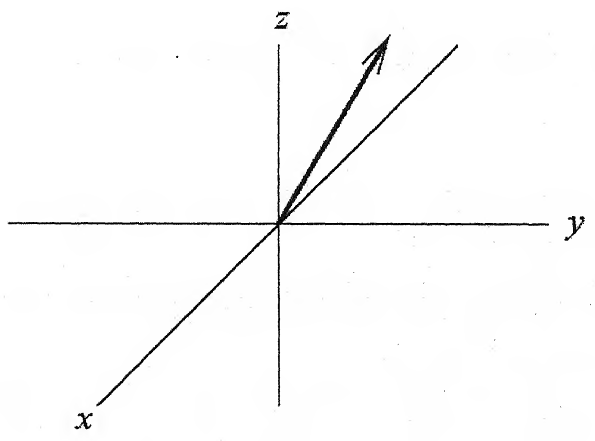


Figure 1.1: Graphical representation of a vector in three dimensions.

A scalar has only a magnitude. Examples of scalar quantities are mass, time and speed.

1.4.1 Vector Algebra

Two vectors are equal if they have the same magnitude and direction. The negative of a vector, denoted $-a$, is a vector of the same magnitude as a but in the opposite direction. We add two vectors a and b by placing the tail of b at the head of a and defining $a+b$ to be the vector with tail at the origin and head at the head of b as shown in figure 1.2.

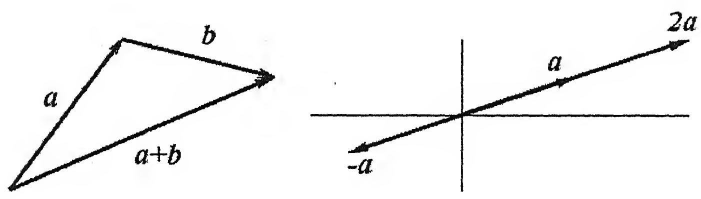


Figure 1.2: Vector arithmetic

The difference, $a-b$, is defined as the sum of a and the negative of b, $a+(-b)$. The result of multiplying a by scalar α is a vector of magnitude $|\alpha||a|$ with the same/opposite direction if α is positive/negative.

Zero and Unit Vectors: The additive identity element for vectors is the zero vector or null vector. This is vector of magnitude zero which is denoted as 0. A unit vector is a vector of magnitude one. If a is nonzero then $a/|a|$ is a unit vector in the direction of a. Unit vectors are often denoted with a caret over-line, $\hat{n}$.

Rectangular Unit Vectors: In n dimensional Cartesian space, the unit vectors in the directions of the coordinates axes are $e_1, e_2, \ldots\ldots e_n$. These are called the rectangular unit vectors.

Linear Dependence Vectors: A set of M-dimensional vectors $\{x_1, x_2, x_3 \ldots\ldots\ldots\ldots x_N\}$ is said to be linearly dependent if there exist numbers $\{c_1, c_2, c_3 \ldots\ldots\ldots\ldots c_N\}$ not all 0 such that

$$c_1x_1 + c_2x_2 + c_3x_3 \ldots\ldots\ldots\ldots\ldots\ldots\ldots + c_Nx_N = 0$$

Inner Product: The inner product of two vectors $x, y \in R^M, x = [x_1, x_2 \ldots\ldots x_M]^T$ *and* $y = [y_1, y_2 \ldots\ldots\ldots\ldots y_M]^T$ is defined as

$$\langle x, y\rangle = \sum_{i=1}^{M} x_i y_i = x^T y \tag{1.1}$$

When the product of two vectors $\langle x, y\rangle$ is 0 , then vectors are said to be orthogonal.

1.4.2 Kronecker Delta and Einstein Summation Convention

The Kronecker Delta tensor is defined as,

$$\delta_{ij} = \begin{cases} 1 & \textit{if } i = j \\ 0 & \textit{if } i \neq j \end{cases} \tag{1.2}$$

This notation is useful in out work with vectors.

Consider writing a vector in terms of its rectangular components. Instead of using ellipses: $a = a_1e_1 + \ldots + a_ne_n$, we could write the expression as a sum: $a = \sum_{i=1}^{n} a_ie_i$, where it is understood that whenever as index is repeated in a term we sum over that index from

1 to n. This is the Einstein summation convention. A repeated index is called a summation index or a dummy index. Other indices can take any value from 1 to n and are called free indices.

1.5 Matrix

A matrix is a rectangular array of values arranged into rows and columns. Here is a matrix A of size m x n:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13}\text{..........}a_{1n} \\ a_{21} & a_{22} & a_{23}\text{........}a_{2n} \\ . & & \\ . & & \\ a_{m1} & a_{m2}\text{...............}a_{mn} & \end{pmatrix}$$

The summation or subtraction of two matrices A and B is defined by adding or subtracting corresponding elements as:

$$A \pm B = \begin{pmatrix} a_{11} & a_{12} & a_{13}\text{..........}a_{1n} \\ a_{21} & a_{22} & a_{23}\text{........}a_{2n} \\ . & & \\ . & & \\ a_{m1} & a_{m2}\text{...............}a_{mn} & \end{pmatrix} \pm \begin{pmatrix} b_{11} & b_{12} & b_{13}\text{..........}b_{1n} \\ b_{21} & b_{22} & b_{23}\text{........}b_{2n} \\ . & & \\ . & & \\ b_{m1} & b_{m2}\text{...............}b_{mn} & \end{pmatrix} = \begin{pmatrix} a_{11} \pm b_{11}\text{..........}a_{1n} \pm b1n \\ \\ .. \\ .. \\ a_{m1} \pm b_{m1}\text{...........}a_{mn} \pm b_{mn} \end{pmatrix}$$

The summation or subtraction is undefined if the size of the matrices is different.

The multiplication of two matrices is possible only when the number of columns of the first matrix is equal to the number of rows of the second matrix i.e. the product for two matrices A and B could be possible only when the order of A is i x j and the order of B must j x k, the product matrix will be of the order of the I x k.

$$(AB)_{i\times k} = \sum_j A_{ij} B_{jk} \tag{1.3}$$

The ***identity matrix*** I has elements I_{ij} such as: $I_{ij} = 1$ when $i = j$ and $I_{ij} = 0$ otherwise.

The ***transpose*** of a matrix $A = \lfloor a_{ij} \rfloor_{m\times n}$ is defined as

$$A^T = \left[a_{ji}\right]_{n\times m} \tag{1.4}$$

The ***inverse*** of a square matrix $A = \lfloor a_{ij} \rfloor_{n\times n}$ is defined as

$$A^{-1} = \frac{Adj(A)}{\det(A)} \tag{1.5}$$

The ***rank*** of a matrix $A \in R^{N\times M}$ is defined as the number of linearly independent rows and columns of *A*. If *A* is a full rank, then its rank is *N* or *M*, which ever is lower.

The ***trace*** of a matrix is defined as a sum of the diagonal elements of the matrix

$$trace(A) = \sum_{i=1}^{n} a_{ii} \tag{1.6}$$

1.5.1 Pseudo inverse

The pseudo inverse $A^\dagger$, also called the *Moore–Penrose generalized inverse*, of a matrix $A \in R_{m\times n}$ is unique, which satisfies

$$AA^\dagger A = A$$

$$A^\dagger AA^\dagger = A^\dagger$$

$$\left(AA^\dagger\right)^T = AA^\dagger$$

$$\left(A^\dagger A\right)^T = A^\dagger A$$

$A^{\dagger}$ can be calculated by

$$A^{\dagger} = (A^{T}A)^{-1}A^{T} \tag{1.7}$$

If $A^{T}A$ is nonsingular, and

$$A^{\dagger} = A^{T}(AA^{T})^{-1} \tag{1.8}$$

if AA^{T} is nonsingular. The pseudo inverse is directly associated with linear LS problems.

When **A** is a square nonsingular matrix, the pseudo inverse $A^{\dagger}$ is its inverse A^{-1} . For a scalar α , if $\alpha \neq 0$, $\alpha^{\dagger} = \alpha^{-1}$; if $\alpha = 0, \alpha^{\dagger} = 0$

1.5.2 Vector norms and Matrix Norms

A norm acts as a measure of distance. A ***vector norm*** on R^n is a mapping $f : R^n \to R$ that satisfies such properties: For any $x, y \in R^n, a \in R$,

- $f(x) \geq 0$, and $f(x) = 0, \mathit{iff}\ x = 0$
- $f(x+y) \leq f(x) + f(y)$.
- $f(ax) = |a| f(x)$.

The mapping is denoted as $f(x) = \|x\|$.

The *p*-norm or L_p-norm is a popular class of vector norms:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{1.9}$$

with $p \geq 1$. Usually, the L_1, L_2 and L_∞ norms are more useful

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \tag{1.10}$$

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} = \left(x^T x\right)^{\frac{1}{2}} \tag{1.11}$$

$$\|x\|_\infty = \max_{1 \le i \le n} |x_i| \tag{1.12}$$

The L_2 -norm is the popular Euclidean norm.

A ***matrix norm*** is generalization of the vector norm by extending R^n to $R^{m \times n}$.

For a matrix **A** = $\lfloor a_{ij} \rfloor_{m \times n}$, the most frequently used matrix norms are the Frobenius norm

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \tag{1.13}$$

And the matrix p-norm

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p \tag{1.14}$$

where sup is the supreme operation.

The matrix 2-norm and the Frobenius norm are invariant with respect to orthogonal transforms, that is, for all orthogonal Q_1 and Q_2 of appropriate dimensions

$$\|Q_1 A Q_2\|_F = \|A\|_F \tag{1.15}$$

$$\|Q_1 A Q_2\|_2 = \|A\|_2 \tag{1.16}$$

1.6 Decomposition

1.6.1 Eigen-value Decomposition

Given a square matrix $A \in R^{n \times n}$, if there exists a scalar λ and a nonzero vector **v** such that

$$Av = \lambda v \tag{1.17}$$

then λ and **v** are, respectively, called an *eigen-value* of **A** and its corresponding *eigenvector*. All the eigen-values λ_i, $i = 1, \cdot \cdot \cdot, n$, can be obtained by solving the characteristic equation :

$$\det(A - \lambda I) = 0 \tag{1.18}$$

where **I** is an $n \times n$ identity matrix. The set of all the eigen-values is called the *spectrum* of **A**.

If **A** is nonsingular, $\lambda_i \neq 0$. If **A** is symmetric, then all the λ_i's are real.

The maximum and minimum eigen-values satisfy the Rayleigh quotient

$$\lambda_{\max}(A) = \max_{v \neq o} \frac{v^T A v}{v^T v} \tag{1.19}$$

The trace of a matrix is equal to the sum of all its eigen-values and the determinant of a matrix is equal to the product of its eigen-values:

$$tr(A) = \sum_{i=1}^{n} \lambda_i \tag{1.20}$$

$$|A| = \prod_{i=1}^{n} \lambda_1 \tag{1.21}$$

1.6.2 Singular Value Decomposition

For a matrix $A \in R^{m\times n}$, there exist real unitary matrices $U = [u_1, u_2, u_3 ....u_m] \in R^{m\times n}$ and $V = [v_1, v_2, v_3 ....v_m] \in R^{m\times n}$ such that

$$U^T A V = \Sigma \tag{1.22}$$

where $\Sigma \in R^{m\times n}$ is a real pseudodiagonal *m* x *n* matrix with $\sigma_i, i = 1,2,3,4...........p,$ $p = \min(m,n), \sigma_1 \ge \sigma_2 \ge \sigma_3 \ge ....\sigma_p \ge 0$, on the diagonal and zeros off the diagonal. σ_i 's are called the singular values of A, u_i and v_i are, respectively called the left singular vector and right singular vector for σ_i .They satisfy the relations:

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^T u_i = \sigma_i v_i .$$

Accordingly A can be written as

$$A = U\Sigma V^T = \sum_{i=1}^{r} \lambda_i u_i v_i^T \tag{1.23}$$

where *r* is the cardinality of the smallest nonzero singular value. In the special case when A is a symmetric non-negative definite matrix, $\Sigma = diag\left(\lambda_1^{\frac{1}{2}}, ...\lambda_p^{\frac{1}{2}}\right)$,

where $\lambda_1 \ge \lambda_2 \ge ...\lambda_p \ge 0$ are the real eigenvalues of **A**,**v***i* being the corresponding eigenvectors.

The SVD is useful in many situations. The rank of **A** can be determined by the number of nonzero singular values. The power of **A** can be easily calculated by

$A^k = U\Sigma^k V^T$, where *k* is a positive integer.

The SVD is extensively applied in linear inverse problems. The pseudoinverse of **A** can then be described by equation (1.7).

The Frobenius norm can thus be calculated as

$$\|A\|_F = \left(\sum_{i=1}^{p} \sigma_i^2 \right)^{\frac{1}{2}} \tag{1.24}$$

and the matrix 2-norm is calculated by

$$\|A\|_2 = \sigma_1 \tag{1.25}$$

1.6.3 QR Decomposition

Although the quantity $(A^T A)^{-1}$ exists, significant numerical difficulties may occur in computing this inverse in instance where $A^T A$ is nearly singular. For the full-rank or over determined linear LS case, $m \geq n$, can also be solved by using QR decomposition procedure.

A is first factorized as

$$A = QR \tag{1.26}$$

where Q is an $m \times m$ orthogonal matrix, that is, $Q^T Q$ = I, and $R = \begin{bmatrix} \bar{R} \\ 0 \end{bmatrix}$ is an $m \times n$ upper triangular matrix with $\bar{R} \in R^{m \times n}$.

Inserting (1.26) into the set of linear equation (SLE) $Ax = b$ and premultiplying by Q^T, we have

$$Rx = Q^T b \tag{1.27}$$

Denoting $Q^T b = \begin{bmatrix} \bar{b} \\ \tilde{b} \end{bmatrix}$, where $\bar{b} \in R^n$ *and* $\tilde{b} \in R^{m-n}$, we have

$$\bar{R}x = \bar{b} \tag{1.28}$$

Since $\bar{R}$ is a triangular matrix, x can be easily solved using backward substitution. This is the procedure used in the GSO procedure.

When rank (A) < n, the rank-deficient LS problem has an infinite number of solutions, the QR decomposition does not necessarily produce an othonormal basis for range(**A**) = $\{y \in R^m : y = Ax \ \mathit{for\ some}\ x \in R^n\}$. The QR decomposition can be applied to produce an orthonormal basis for range (**A**).

The QR decomposition is a basic method for computing the SVD. The QR decomposition itself can be computed by means of the Givens rotation, the Householder transform, or the GSO.

1.6.4 Condition Numbers

The condition number of a matrix $\mathrm{A} \in \mathrm{R}^{\mathrm{m} \times \mathrm{n}}$ is defined by

$$cond_p(\mathrm{A}) = \|\mathrm{A}\|_p \|\mathrm{A}^{\dagger}\|_p \tag{1.29}$$

where p can be selected as 1, 2, ∞, Frobenius, or any other norm. The relation, cond(**A**) ≥ 1, always holds. Matrices with small condition numbers are well conditioned, while matrices with large condition number are poorly conditioned or ill-conditioned. The condition number is especially useful in numerical computation, where ill-conditioned matrices are sensitive to rounding errors. For the L_2–norm,

$$cond_2(\mathrm{A}) = \frac{\sigma_1}{\sigma_2} \tag{1.30}$$

where $p = \min(m,n)$.

1.7 Optimization Methods

1.7.1 Vector Gradient

Let g be a differentiable scalar function of m variables,

$$g = g(w_1,\dots\dots,w_n) = g(\mathbf{w}) \tag{1.31}$$

where $\mathbf{w} = (w_1,\dots\dots,w_m)^T$. Then the vector gradient of g w.r.t. $\mathbf{w}$ is the m-dimensional vector of partial derivatives of g,

$$\frac{\partial g}{\partial \mathbf{w}} = \nabla g = \nabla \mathbf{w} g = \begin{pmatrix} \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_m} \end{pmatrix} \tag{1.32}$$

Similarly, we can define second-order gradient or Hessian matrix,

$$\frac{\partial^2 g}{\partial \mathbf{w}^2} = \begin{pmatrix} \frac{\partial^2 g}{\partial w_1^2} & \cdots & \frac{\partial^2 g}{\partial w_1 w_m} \\ \vdots & & \vdots \\ \frac{\partial^2 g}{\partial w_m w_1} & \cdots & \frac{\partial^2 g}{\partial w_m^2} \end{pmatrix} \tag{1.33}$$

Generalization to the vector valued functions $g(\mathbf{w}) = (g_1(\mathbf{w}),\dots\dots,g_n(\mathbf{w}))^T$ leads to a definition of the Jacobian matrix of g w.r.t. $\mathbf{w}$,

$$\frac{\partial g}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial g_1}{\partial w_1} & \cdots & \frac{\partial g_n}{\partial w_1} \\ \vdots & & \vdots \\ \frac{\partial g_1}{\partial w_m} & \cdots & \frac{\partial g_n}{\partial w_m} \end{pmatrix} \tag{1.34}$$

In this vector convention, the columns of the Jacobian matrix are gradients of the corresponding components functions g_i w.r.t. the vector $\mathbf{w}$.

The differentiation rules are analogous to the case of ordinary functions:

$$\frac{\partial f(\mathbf{w})g(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} g(\mathbf{w}) + f(\mathbf{w}) \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \tag{1.35}$$

$$\frac{\partial f(\mathbf{w}) / g(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} g(\mathbf{w}) - f(\mathbf{w}) \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \right] \Big/ g^2(\mathbf{w}) \tag{1.36}$$

$$\frac{\partial f(g(\mathbf{w}))}{\partial \mathbf{w}} = f'(g(\mathbf{w})) \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \tag{1.37}$$

1.7.2 Matrix Gradient

Consider a scalar valued function g of the $n \times m$ matrix $\mathbf{W} = (w_{ij})$ (e.g. determinant of a matrix). The matrix gradient w.r.t $\mathbf{W}$ is a matrix of the same dimension as $\mathbf{W}$ consisting of partial derivatives of g w.r.t. components of $\mathbf{W}$:

$$\frac{\partial g}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial g}{\partial w_{11}} & \cdots & \frac{\partial g}{\partial w_{1n}} \\ \vdots & & \vdots \\ \frac{\partial g}{\partial w_{m1}} & \cdots & \frac{\partial g}{\partial w_{mn}} \end{pmatrix} \tag{1.38}$$

1.7.3 Taylor Expansion of Multivariate Functions

The well known formula for Taylor series expansion of a scalar function $g(w)$ reads,

$$g(w') = g(w) + \frac{dg}{dw}(w'-w) + 1/2 \frac{d^2 g}{dw^2}(w'-w)^2 + \ldots = g(w) + \sum g^i(w)(w'-w)^i \tag{1.39}$$

This can be generalized to the function of m variables

$$g(\mathbf{w}')=g(\mathbf{w})+\frac{\partial g}{\partial \mathbf{w}}(\mathbf{w}'-\mathbf{w})+1/2(\mathbf{w}'-\mathbf{w})^T\frac{\partial^2 g}{\partial \mathbf{w}^2}(\mathbf{w}'-\mathbf{w})+\ldots \tag{1.40}$$

with derivatives evaluated at $\mathbf{w}$. Note that the second term is an inner product of a gradient of g with the vector $\mathbf{w}'-\mathbf{w}$ and the third term is a quadratic form with the Hessian matrix $\frac{\partial^2 g}{\partial \mathbf{w}^2}$.

Similarly for a scalar function of a matrix variable:

$$g(\mathbf{W}')=g(\mathbf{W})+trace\left[\left(\frac{\partial g}{\partial \mathbf{W}}\right)^T(\mathbf{W}'-\mathbf{W})\right]+\ldots \tag{1.41}$$

Reminder: $$trace(\mathbf{A})=\sum_{i=1}^{n}a_{ii} \tag{1.42}$$

i.e. trace is defined as a sum of the diagonal elements of the matrix. The above formula shows 2 terms of Taylor expansion. It uses the extension of definition of an inner product to matrices,

$$\mathbf{AB}=\sum_{i=1}^{m}\sum_{j=1}^{m}a_{ij}b_{ij} \tag{1.43}$$

but

$$trace(\mathbf{A}^T\mathbf{B})=\sum_{i=1}^{m}(\mathbf{A}^T\mathbf{B})_{ii}=\sum_{i=1}^{m}\sum_{j=1}^{m}a_{ij}b_{ij} \tag{1.44}$$

1.7.4 Unconstrained Optimization

Gradient descent is a method of minimization of a given cost or objective function $J(\mathbf{w})$:

- Start at some initial point $\mathbf{w}(0)$;
- Calculate gradient of $J(\mathbf{w})$ at $\mathbf{w}(0)$;
- Move in the direction of the negative gradient or steepest descent by some distance;
- Repeat above until consecutive points are sufficiently close.

In mathematical notation the above procedure reads

$$\Delta\mathbf{w}(t) = -\alpha(t)\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \tag{1.45}$$

or

$$\Delta\mathbf{w}\,\alpha - \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \tag{1.46}$$

- Gradient descent moves always downwards in a hilly landscape
- Local minima can trap the movement
- Initialization is important to avoid local minima
- Choice of the learning rate is crucial for speed of convergence;

Stochastic gradient descent

- Data dependent cost functions;

- Statistical model for data;
- Typical form of cost function $J(\mathbf{w}) = E[g(\mathbf{w}, \mathbf{x})]$, where E denotes expectation (i.e. $E[X] = \int x \int(x)dx$);
- $\mathbf{x}$ is the random vector modeling observation vector;
- (Unknown) pdf of $\mathbf{x}$ is $f(\mathbf{x})$ w.r.t. which the expectation is taken;
- Usually only a sample $\mathbf{x}(1), \mathbf{x}(2), \ldots$ is given

The steepest descent-learning rule becomes,

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \alpha(t)\frac{\partial}{\partial \mathbf{w}} E[g(\mathbf{w}, \mathbf{x}(t))]|_{\mathbf{w}=\mathbf{w}(t-1)} \tag{1.47}$$

or

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \alpha(t)\frac{\partial}{\partial \mathbf{w}} \int g(\mathbf{w}, \xi) f(\xi) d\xi \tag{1.48}$$

For twice differentiable $g(\mathbf{w}, \mathbf{x})$ w.r.t. $\mathbf{w}$ this equals,

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \alpha(t)\int\left[\frac{\partial}{\partial \mathbf{w}} g(\mathbf{w}, \xi)\right] f(\xi) d\xi \tag{1.49}$$

- Approximate the expectation by the sample mean
- Batch-learning: using all available data in each step

On line version is possible - drop the expectation,

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \alpha(t)\frac{\partial}{\partial \mathbf{w}} g(\mathbf{w}, \xi)|_{\mathbf{w}=\mathbf{w}(t-1)} \tag{1.50}$$

- Calculates direction of next move using only 1 data point (incoming)
- The direction of movement fluctuates highly between the steps but
- The average direction is approximately the steepest descent of the batch version
- Lower computational cost
- Slower - needs more iterations to converge

1.8 Dynamical Systems

For a dynamical system described by a set of ordinary differential equations, the stability of the system can be examined by Lyapunov's second theorem or the Lipschitz condition.

Lyapunov's second theorem: For a dynamic system described by a set of differential equations

$$\frac{dx}{dt} = f(x) \tag{1.51}$$

where x = $(x_1(t), x_2(t), \cdot \cdot, x_n(t))^T$ *and* f = $(f_1, f_2, \cdot \cdot, f_n)^T$. There exists a positive definite function $E = E(\mathbf{x})$, called a Lyapunov function or energy function, *so* that

$$\frac{dE}{dt} = \sum_{j=1}^{n} \frac{\partial E}{\partial x_j} \frac{\partial x_j}{dt} \le 0 \tag{1.52}$$

with $\frac{dE}{dt} = 0$ only for $\frac{dx}{dt} = 0$, then the system is stable, and the trajectories $\boldsymbol{x}$ will asymptotically converge to stationary points as $t \to \infty$.

The stationary points are also known as equilibrium points and attractors. The crucial step in applying the Lyapunov's second theorem is to find a suitable energy function.

Lipschitz Condition: For a dynamic system described by equation(1.51) , a sufficient condition that guarantees the existence and uniqueness of the solution is given by the Lipschitz condition

$$\|f(x_1)-f(x_2)\| \leq \gamma \|x_1 - x_2\| \tag{1.53}$$

where γ is any positive constant, called the Lipschitz constant, and $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ are any two variables in the domain of the function vector **f** . **f** (**x**) is said to be Lipschitz continuous. If $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are in some neighborhood of $\boldsymbol{x}$, then they are said to satisfy the Lipschitz condition locally and will reach a unique solution in the neighborhood of $\boldsymbol{x}$. The unique solution is a trajectory that will converge to an attractor asymptotically and reach it only at $t \rightarrow \infty$.

1.9 Probability

Definition:

Ω : Sample space; contains all possible outcomes of an experiment. Ω can be discrete or continuous.

ω : A single outcome; $\omega \in \Omega$

A: Specific event of interest or set of outcomes $A \subseteq \Omega$. An event A is said to occur if the observed outcome ω is an element of A, that is , $\omega \in A$. Associated that with these definitions is a probability space.

For example, $A_1 \subseteq A_2 \Rightarrow P(A_1) \leq P(A_2)$

where $P(\mathrm{A})$ is a probability function that assigns a real, scaler-valued function to each set A, with the constraints:

1. $P(\mathrm{A}) \geq 0 \quad \forall \quad \mathrm{A} \in \Omega$

2. $P(\Omega) = 1$

3. If sets or experiments outcomes $\mathrm{A}_1, \mathrm{A}_2 \ldots\ldots\ldots\ldots\ldots\ldots \mathrm{A}_n$ are mutually exclusive, that is, $\mathrm{A}_i \cap \mathrm{A}_j = \phi, \quad \forall i, j, then \; P\left(\bigcup_{i=1}^{n} \mathrm{A}_i\right) = \sum_{i=1}^{n} p(\mathrm{A}_i)$.

   Given the set of all outcomes $\mathrm{A}_i, i = 1,2,3,4 \ldots\ldots\ldots\ldots\ldots n$, which constitute a partition of Ω, $\quad \sum_i P(\mathrm{A}_i) = 1$

4. $P(\phi) = 0$

1.10 Random Variables

Assume we are conducting an experiment that involves asking 10 people whether they like ice cream or not. For each of the surveyed people we record '1' if they say they like it, '0' if they say they don't. The sample space for this experiment is the space of all binary strings of length 10, i.e., it has 2^{10} elements. It follows that there are $2^{2^{10}}$ possible events in this sample space. If we are only interested in how many said that they liked ice cream, then we can reduce the sample space to the numbers 0 through 10 which is easier to deal with than the original space. Note that we really defined a function from the sample space into the numbers 0, 1, . . .,10 by counting the number of '1''s in the binary string encoding of the answers. In general, we define a *random variable* to be a function from a sample space into the real numbers. In most experiments, random variables are used implicitly as in "sum of numbers" in a toss of two dice, "number of heads" in 25 coin tosses, etc. Note that in defining a random variable we have defined a new sample space as the range of the variable, {0, 1, . . . , 10} in our ice cream experiment. Given a sample space with a probability function P and we define a random variable X, we can define a probability function P_X for the range of X using P the following way. Let X be the range of X, and let $x \in X$. We observe $X = x$ if and only if the outcome of the experiment is an $s \in S$ such that $X(s) = x$. Hence,

$$P_X(X = x) = P(\{s \in S \mid X(s) = x\}) \tag{1.54}$$

If we concern with event $A \subseteq X$, we define,

$$P_X(A) = P(\{s \in S \mid X(s) \in A\}) \tag{1.55}$$

We often write $P(X \in A)$ for $P_X(A)$ and $P(X = x)$ for $P_X(X = x)$ if no confusion can arise.

We can associate several functions with a random variable. The first is the cumulative distribution function or CDF, defined by,

$$F_X(x) = P_X(X \le x), \text{ for all } x \tag{1.56}$$

A random variable is discrete if $F_X(x)$ is a step function of x, and is continuous if $F_X(x)$ is continuous. The probability mass function or PMF of a discrete random variable is defined as,

$$f_X(x) = P_X(X = x), \text{ for all } x \tag{1.57}$$

Similarly, the probability density function of a continuous random variable is defined as the function f_X that satisfies,

$$F_X(x) = \int_{-\infty}^{x} f_X(t)dt, \text{ for all } x \tag{1.58}$$

1.11 Probability Distribution and Densities

The Probability Distribution

The probability distribution $f(x)$ of a discrete random variable is defined as,

$$P(X = x) = f(f) \tag{1.59}$$

The Cumulative Distribution Function (CDF)

The CDF of a discrete random variable is,

$$F_x(x_0) = \sum_{u \le x_0} f(u) \tag{1.60}$$

The CDF $f(x)$ of a random vector $\mathbf{x} = (x_1, x_2, \ldots\ldots, x_n)^T$ at point $x = x_0$,

$$F_x(x_0) = P(x \le x_0) \tag{1.61}$$

The Multivariate Probability Density Function (PDF)

The PDF $p_x(x)$ of a continuous random vector x is defined as,

$$p_x(x_0) = \frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2}\ldots\ldots\frac{\partial}{\partial x_n} F_x(x)\,|_{x=x_0} \tag{1.62}$$

Hence,

$$F_x(x_0) = \int_{-\infty}^{x_0} p_x(x)dx \tag{1.63}$$

For discrete random variable the corresponding formula is,

$$F_x(x_0) = \sum_{u \le x_0} f(u) \tag{1.64}$$

where the sum is taken over all u's, for which $u \le x_0$.

The Joint Distribution Function

The joint distribution function of vectors x and y is given by,

$$F_{x,y}(x_0, y_0) = P(x \le x_0, y \le y_0) \tag{1.65}$$

where x_0, y_0 are vectors of dimensions of x and y, respectively. Thus, the joint distribution function calculates the probability of the event $x \le x_0$ and $y \le y_0$.

The *joint density function* is defined analogously to previous definitions by differentiation of the joint probability distribution w.r.t. all components. It follows that the probability of an event $(x \le x_0, y \le y_0)$ is,

$$P(x \le x_0, y \le y_0) = \int_{-\infty}^{x_0} \int_{-\infty}^{y_0} p_{x,y}(\xi, \eta) d\eta d\xi \tag{1.66}$$

1.12 Gaussian Distribution

The Gaussian distribution, known as the *normal distribution*, is the most common assumption for error distribution. The PDF of the normal distribution is defined as

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1.67}$$

for $x \in R$, where μ is the mean and $\sigma > 0$ is the standard deviation. For the Gaussian distribution, 99.73% of the data are within the range of $[\mu - 3\sigma, \mu+3\sigma]$. The Gaussian distribution has its first-order moment as μ, second order moment as σ^2, and higher-order moments as zero. If $\mu = 0$ and $\sigma = 1$, the distribution is called the *standard normal distribution.* The PDF is also known as the *likelihood function.* An ML estimator is a set of values (μ, σ) that maximizes the likelihood function for a fixed value of x.

The cumulative distribution function (CDF) is defined as the probability that a random variable is less than or equal to a value x, that is

$$F(x) = \int_{-\infty}^{x} p(t)dt \tag{1.68}$$

The standard normal CDF, conventionally denoted Φ, is given by setting $\mu = 0$ and $\sigma = 1$. The standard normal CDF is usually expressed by

$$\Phi(x) = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right] \tag{1.69}$$

where the error function erf *(x)* is a non elementary function, which is defined by

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{1.70}$$

When vector $x \in R^n$, the PDF of the normal distribution is then defined by

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{1.71}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean vector and the covariance matrix, respectively.

1.13 Cauchy Distribution

The Cauchy distribution, also known as the *Cauchy–Lorentzian distribution,* is another popular data-distribution model. The density of the Cauchy distribution is defined as

$$p(x) = \frac{1}{\pi\sigma\left[1+\left(\frac{x-\mu}{\sigma}\right)^2\right]} \tag{1.72}$$

for $x \in R$, where μ specifies the location of the peak and σ is the scale parameter that specifies the half-width at the half-maximum. When $\mu = 0$ and $\sigma = 1$, the distribution is called the *standard Cauchy distribution.*

Accordingly, the CDF of the Cauchy distribution is calculated by

$$F(x) = \frac{1}{\pi}\arctan\left(\frac{x-\mu}{\sigma}\right)+\frac{1}{2} \tag{1.73}$$

None of the moments is defined for the Cauchy distribution. The median of the distribution is equal to μ. The Cauchy distribution has a longer tail than the Gaussian distribution, and this makes it more valuable in stochastic search algorithms by searching larger subspaces in the data space.

1.14 Markov Processes, Markov Chains, and Markov-chain Analysis

Given a stochastic process $\{X(t): t \in \mathrm{T}\}$, where *t* is time, *X*(*t*) is a state in the state space *S*.

A Markov process is defined as a stochastic process that satisfies the relation characterized by the conditional distribution

$$\begin{aligned} &P\left[X(t_0+t_1) \le x \middle| X(t_0) = x_0, X(\tau) = x_\tau, -\infty < \tau < t_0\right] \\ &\quad = P\left[X(t_0+t_1) \le x \middle| X(t_0) = x_0\right] \end{aligned} \tag{1.74}$$

for any value of t_0 and for $t_1 > 0$. The future distribution of the process is determined by the present value of $X(t_0)$ only.

When T and S are discrete, a Markov process is called a *Markov chain.* Conventionally, time is indexed using integers, and a Markov chain is a set of random variables that satisfy

$$\begin{aligned} &P[X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2} \ldots] \\ &= P[X_n = x_n | X_{n-1} = x_{n-1}] \end{aligned} \tag{1.75}$$

This definition can be extended for multistep Markov chains, where a chain state has conditional dependency on only a finite number of its previous states.

For a Markov chain, $P[X_n = j | X_{n-1} = i]$ is the transition probability of state i to j at time n - 1. If

$$P[X_n = j | X_{n-1} = i] = P[X_{n+m} = j | X_{n+m-1} = i] \tag{1.76}$$

for $m \geq 0$ and $i, j \in S$, the chain is said to be *time homogeneous*. In this case, one can denote

$$P_{i,j} = P[X_n = j | X_{n-1} = i] \tag{1.77}$$

and the transition probabilities can be represented by a matrix, called the *transition matrix*, $\mathbf{P} = [P_{i,j}]$, where $i, j = 0, 1, \cdot \cdot \cdot$. For finite S, $\mathbf{P}$ has a finite dimension.

In the Markov-chain analysis, the transition probability after k step transitions is $\mathbf{P}^k$. The *stationary distribution* or *steady-state distribution* is a vector that satisfies

$$\mathbf{P}^{\mathrm{T}} \pi^* = \pi^* \tag{1.78}$$

That is, π^* is the left eigenvector of **P** corresponding to the eigenvalue 1. If **P** is irreducible and aperiodic, that is, every state is accessible from every other state and in the process none of the states repeats itself periodically, then $\mathbf{P}^k$ converges element wise to a matrix each row of which is the unique stationary distribution π^*, with

$$\lim_{k\to\infty}\left(P^K\right)^T \pi = \pi^* \tag{1.79}$$

Many modeling applications are Markovian, and the Markov-chain analysis is widely used for convergence analysis for algorithms.

1.15 Bayes Decision Rule for Minimum Error

Consider C classes, $\omega_1, \omega_2, \omega_3 \ldots\ldots\ldots\ldots \omega_C$, with *a priori* probabilities (the probabilities of each class occurring) $p(\omega_1), P(\omega_2), \ldots\ldots\ldots\ldots p(\omega_C)$, assumed known. If we wish to minimize the probability of making an error and we have no information regarding an object other than the class probability distribution then we would assign an object to class ω_j if

$$p(\omega_j) > p(\omega_k) \quad k = 1\ldots\ldots C;\ k \neq j \tag{1.80}$$

This classifies all objects as belonging to one class. For classes with equal probabilities, patterns are assigned arbitrarily between those classes.

However, we do have an *observation vector* or *measurement vector* $\boldsymbol{x}$ and we wish to assign $\boldsymbol{x}$ to one of the C classes. A decision rule based on probabilities is to assign $\boldsymbol{x}$ to class ω_j if the probability of class ω_j given the observation $\boldsymbol{x}$, $p(\omega_j|x)$, is greatest over all classes $\omega_1, \omega_2, \omega_3 \ldots\ldots\ldots\ldots \omega_C$. That is, assign $\boldsymbol{x}$ to class ω_j if

$$p(\omega_j|x) > p(\omega_k|x) \quad k = 1\ldots\ldots\ldots\ldots C; k \neq j \tag{1.81}$$

This decision rule partitions the measurement space into C regions $\Omega_1 \ldots\ldots\ldots\ldots\ldots \Omega_C$ such that if $x \in \Omega_j$ then x belongs to class ω_j.

The *a posteriori* probabilities $p(\omega_j|x)$ may be expressed in terms of the *a priori* probabilities and the class-conditional density functions $p(x|\omega_j)$ using Bayes' theorem as

$$p(\omega_i|x) = \frac{p(x|\omega_i)p(\omega_1)}{p(x)} \qquad (1.82)$$

and so the decision rule (1.81) may be written: assign x to ω_j if

$$p(x|\omega_j)p(\omega_j) > p(x|\omega_k)p(\omega_k) \qquad k = 1,\ldots\ldots\ldots\ldots C; k \neq j \qquad (1.83)$$

This is known as Bayes' rule for *minimum error*.

For two classes, the decision rule (1.82) may be written

$$l_r = \frac{p(x|\omega_1)}{p(x|\omega_2)} > \frac{p(\omega_2)}{p(\omega_1)} \text{ implies } x \in class\ \omega_1 \qquad (1.84)$$

The fact that the decision rule (1.82) minimizes the error may be seen as follows. The probability of making an error, *p(*error), may be expressed as

$$p(error) = \sum_{i=1}^{C} p(error|\omega_i)p(\omega_i) \qquad (1.85)$$

where $p(error|\omega_i)$ is the probability of misclassifying patterns from class ω_i. This is given by

$$p(error|\omega_i) = \left| \int_{C[\Omega_i]} p(x|\omega_i)dx \right. \qquad (1.86)$$

the integral of the class-conditional density function over $C[\Omega_i]$, the region of measurement space outside Ω_i (C is the complement operator), i.e. $\sum_{j=1, j\neq i}^{C} \Omega_j$

. Therefore, we may write the probability of misclassifying a pattern as

$$\begin{aligned} p(error) &= \sum_{i=1}^{C} \int_{C[\Omega_i]} p(x|\omega_i)p(\omega_i)dx \\ &= \sum_{i=1}^{C} p(w_i)\left(1 - \int_{\Omega_i} p(x|\omega_i)dx\right) \\ &= 1 - \sum_{i=1}^{C} p(\omega_i)\int_{\Omega_i} p(x|\omega_i)dx \end{aligned} \tag{1.87}$$

from which we see that minimizing the probability of making an error is equivalent to maximizing

$$\sum_{i=1}^{C} p(\omega_i)\int_{\Omega_i} p(x|\omega_i)dx \tag{1.88}$$

the probability of correct classification. Therefore, we wish to choose the regions Ω_i so that the integral given in (1.86) is a maximum. This is achieved by selecting Ω_i to be the region for which $p(\omega_i)p(x|\omega_i)$ is the largest over all classes and the probability of correct classification, c, is

$$C = \int \max_i p(\omega_i)p(x|\omega_i)dx \tag{1.89}$$

where the integral is over the whole of the measurement space, and the Baye's error is

$$e_B = 1 - \int \max_i p(\omega_i)p(x|\omega_i)dx \tag{1.90}$$

References:

[1] Stuart, R., and Norvig, P., “Artificial Intelligence: A Modern Approach”, 2nd Edition, Pearson, Prentice Hall,(2005)

[2] Schalkoff, R., “Patten Recognition: Statistical, Structural and Neural Network.” Wiley India, ISBN 978-81-265-1370-3, (2007).

[3] Yegnarayana, B., “Artificial Neural Networks”, Prentice Hall of India, (2004).

[4] Haykin, S., “Neural Networks”, Second Edition, Pearson Education (Singapore), (2004).

[5] Du K.L. and Swamy, M.N.S., “Neural Networks in a Soft computing Framework”, Springer-Verlag London Limited, (2006).

[6] Kreyszig, E., “Advanced Engineering Mathematics.”, Wiley, 8th Edition, ISBN 978-0471154969,(1998).

[7] Webb, A.R., “Statistical Pattern Recognition.”, John Wiley and Sons Ltd., 2nd, ISBN 0-470-84514-7,(2002).

CHAPTER 2

Pattern Recognition: Scope and Methods

Chapter 2: Pattern Recognition: Scope and Methods

Abstract

Pattern recognition encompasses a wide range of information processing problem of great practical significance, form speech recognition and the classification of hand written characters, to fault detection in machinery and medical diagnosis. It has been well understood that these problem can perform well and effortlessly by human brain. However, their solution using computer has, in many case proved to be immensely difficult. In order to have the best opportunity of developing effective solutions, it is important to consider the various existing approaches and methods for pattern recognition by the machine. In this chapter we are discussing the different approaches of pattern recognition starting from conventional statistical inference to modern approaches of soft computing.

2.1 Introduction

Recognition is a basic property of all human beings; when a person sees an object, he or she first gathers all information about the object and compares its properties and behaviors with the existing knowledge stored in the mind. If we find a proper match, we recognize it [1]. The concept of recognition is simple in the real world environment, but in the world of computer science, recognizing any object is an amazing feat. The functionality of the human brain is amazing; it is not comparable with any machines or software. The act of recognition can be divided into two broad categories:

1. Concrete item recognition, it involves the recognition of spatial samples such as fingerprints, weather maps, pictures and physical objects and the recognition of temporal samples such as, waveforms and signatures.
2. Abstract item recognition, it involves the recognition of a solution to a problem, an old conversation or argument.

Pattern recognition, as a subject, spans a number of scientific disciplines, uniting them in search for a solution to the common problem of recognizing the pattern of a given class and assigning the name of identified class.

But what is a pattern? Watanabe [2] defines a pattern as opposite of a chaos; it is an entity, vaguely defined, that could be given a name. Pattern could be a fingerprint image, a handwritten cursive word, a human face, or a speech signal.

For example, some pattern recognition system applications include the following. In weather prediction, input data are in the form of weather maps, and the output is a forecast. The symptoms serve as input data in medical diagnosis while disease identity serves as the output. The predicted market ups and downs are the desired output for stock market prediction when input data are the financial news and charts.

Pattern recognition is the categorization of input data into identifiable classes through the extraction of significant attributes of the data from irrelevant background detail. A pattern class is a category determined by some common attributes. Therefore, a pattern is the description of a category member representing a pattern class. A pattern class is a family of patterns that shares some common properties. Pattern recognition by machine involves techniques for assigning patterns to their classes automatically with as little human interventions is possible. Pattern recognition aims to classify data (patterns) based on either a priori knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space [3].

A complete pattern recognition system consists of a sensor that gathers the observations to be classified or described; a feature extraction mechanism that computes numeric or symbolic information from the observations; and a classification or description scheme that does the actual job of classifying or describing observations, relying on the extracted features.

Pattern recognition problems may be logically divided into two broad categories. First involves the study of pattern recognition capabilities of human beings and other living organism, while the second involves the development of theory and techniques for the design of device capable of performing a given recognition task for a specific applications.

The first area deals with the subjects of psychology, physiology and biology, but the second area deals with the computer, information science and artificial intelligence. The pattern recognition is concerned primarily with description and analysis or classification of measurements taken from physical or mental process [4]. The area of pattern recognition deals with the following:

i. Character Recognition: Optical or hand written character recognition,

ii. Visual Image Recognition,

iii. Voice Data Recognition,

iv. Speech Analysis,

v. Man and Machine Diagnostics,

vi. Person Identification and Industrial Inspection.

During the last few years the researchers have proposed many mathematical approaches to solve the pattern recognition problems. The available methods of pattern recognition may be categorized into three basic principals:

i. Statistical Methods; consisting the sub disciplines like discriminant analysis, feature extraction, error estimation, cluster analysis

ii. Structural Methods consisting grammatical inference and parsing

iii. Artificial Intelligence based Methods

2.2 Statistical Methods

An approach to machine intelligence which is based on statistical modeling of data. In a statistical model, one applies probability theory and decision theory to get an algorithm. During the past there has been progress in theory and applications of 'Statistical Pattern Recognition' [5-9]. The three major issues encountered in the design of a statistical pattern recognition system are sensing, feature extraction, and classification. The primary issue is the representation of the input data which can be measured from the objects to be recognized and it is called sensing problem. Each measured quantity describes the characteristics of the pattern or objects.

The number of features of the pattern samples is usually very large. The features of the pattern samples are reduced by considering their salient characteristics. This process is referred to as feature extraction. Several approaches for feature extraction have been proposed by the various researchers such as feature extraction by moments invariants [10], feature extraction by autoregressive models [11], and feature extraction by KL transformation [12].

The last and the third issue of the statistical pattern recognition is the pattern classification or development of the classifier. The pattern classifier is defined as a device or a process that sorts the given data into identifiable categories and classes. The pattern classification is an information transformation process, i.e., the classifiers transforms relatively large set of mysterious data into a smaller set of useful data [9]. Trainable classifier is one that can improve its performance in response to the information it receives as a function of time. Training is a process by which the parameters of the classifiers are adjusted. The classifier is trained using the reduced pattern samples. It is often assumed that the pattern samples of a given class occupy a finite region in a pattern space and it is called a class region.

A good classification is the main object of recognition system. Generally, the statistically pattern recognition problems fall into two main categories: supervised classification (discriminant analysis) problems and unsupervised classification (clustering) problems. When the samples have known classification (labeled samples) then the classification is called the supervised otherwise it is unsupervised [14,15]. For the supervised classification, the data is labeled; and the problem is to design a classifier to predict the class label for any given sample. For the unsupervised classification, data are not labeled and to classify the data into groups with features that distinguish one group from another.

In the supervised classification, each sample is assumed to come from one of the c possible classes Ω_j, $j=1,2,\ldots\ldots,c$. Suppose, we are given n training samples (x_i, y_i), $i=1,2,\ldots\ldots n$, where $x_i \in R^d$ represents the data and $y_i \in \{1,2,\ldots\ldots,c\}$ the corresponding class labels. The objective is to design a decision function $g(x)$ from these training samples such that $g(x)$ can accurately predict the class label for any given test sample x.

A classical way to tackle this problem is to use Bayes' decision rule [16], which assigns a test sample x to the class with the maximum posterior probability $p(\Omega_j|x)$, that is,

$$x \in \Omega_k, if, p(\Omega_k|x) = \max_{j=1,2,\ldots\ldots c} p(\Omega_j|x) \tag{2.1}$$

where $p(\Omega_j|x)$ is defined as the conditional probability that the observed object belongs to class Ω_j given the associated sample data, x. In practice these posterior probabilities are usually unknown. By applying Bayes' theorem, they may be calculated as,

$$p(\Omega_j|x) = \frac{p(x|\Omega_j)p(\Omega_j)}{p(x)} \tag{2.2}$$

where $p(x)$ is the probability density that the sample x occurs, $p(\Omega_j)$ is the prior probability that class Ω_j occurs and $p(x|\Omega_j)$ is the class-conditional density of a sample

x occurring given that the sample is known to come from class Ω_j. Then the decision rule becomes,

$$x \in \Omega_k \ \mathit{if} \ \ p(x|\Omega_k)p(\Omega_k) \geq p(x|\Omega_j)p(\Omega_j), \forall j = 1,2,\ldots\ldots,c \tag{2.3}$$

Generally, the prior probability $p(\Omega_j)$ and the class-conditional densities $p(x|\Omega_j)$ are unknown and need to be estimated. Compared with the estimation of the prior probability $p(\Omega_j)$, the estimation of the class-conditional densities $p(x|\Omega_j)$ is much more difficult.

If we know the functional form of the class-conditional densities $p(x|\Omega_j)$ (e.g., multivariate Gaussian), but do not know the parameters (e.g., mean and covariance), we face a parametric decision problem. In this case, a common strategy is to replace the unknown parameters in the density functions with some estimated values, resulting in the so-called Bayes plug-in classifier. If the form of the class-conditional densities is not known, then we operate in a non-parametric mode. In this case, we may either directly estimate the density for a given sample (e.g., Parzen windows and k-Nearest Neighbor) or construct a decision boundary based on the training samples. In the latter case, assuming that the decision boundary is linear leads to a linear discriminant method (e.g., perceptrons and Fisher's rule). Assuming that the decision boundary is not linear leads to a nonlinear discriminant method (e.g., feedforward neural networks and support vector machines). These basic approaches can be organized into the tree structure as shown in Figure 2.1.

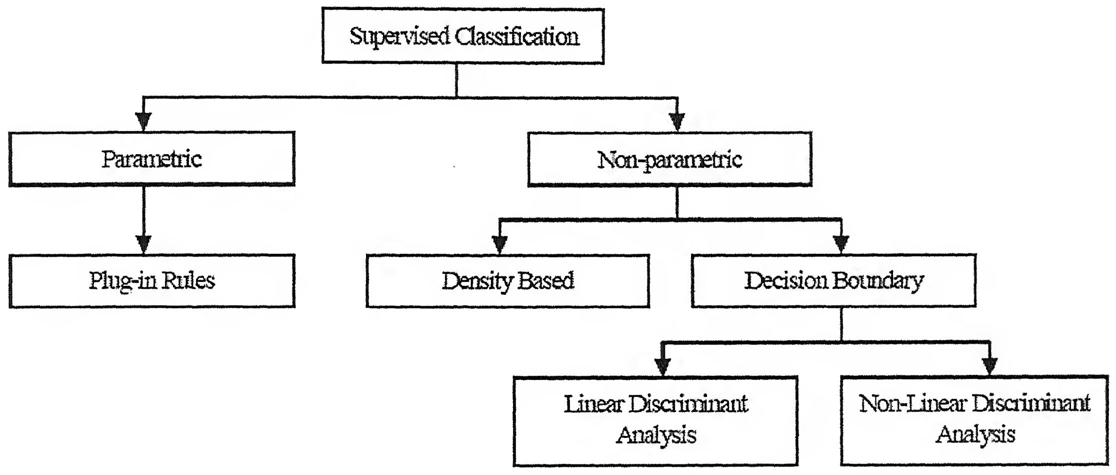

Supervised Classification
Parametric
Non-parametric
Plug-in Rules
Density Based
Decision Boundary
Linear Discriminant Analysis
Non-Linear Discriminant Analysis

Figure 2.1: Various Approaches in Supervised Classification

2.2.1 Parametric Methods

In the parametric approach the underlying densities of the pattern classes are known but the values of the parameters might be unknown. If the parameters were known then the discriminant functions based on them can be readily specified. In practice when large number of pattern samples are available, class density function can be estimated or learned from the samples [4]. These parameters are then used for specification of discriminant functions. An important situation in which pattern classes are characterized by set of parameters occur when the patterns in each of the N classes are random variables governed by N distinct probability functions. The most widely accepted parametric type classifier in pattern recognition is *Bayes classifier.* This classifier assigns zero loss to correct classifications and equal loss to incorrect classifications. The optimal decision of the Bayes classifier minimizes the probability of error in classification. For a two-class problem the Bayes discriminator gives an optimum decision by assigning the pattern sample

If the functional form of the class-conditional densities $p(x|\Omega_j)$ is known, but the parameters of the density functions are unknown, we may estimate the unknown parameters from the available samples. Without loss of generality, we may drop the class label and write $p(x|\Omega_j)$ as $p(x;\theta)$, where θ is a parameter vector to be estimated. The commonly used parametric models are the multivariate Gaussian distribution for continuous variables, the binomial distribution for binary variables, and the multi normal distribution for integer-valued (or categorical) variables.

Applying the most widely used estimator, namely the maximum likelihood estimator, we select as to estimate the particular value of θ, which gives the greatest value of the "likelihood", which is defined by,

$$L(x_1,\ldots\ldots,x_n;\theta)=\prod_{i=1}^{n}p(x_i;\theta) \tag{2.4}$$

An important result of using this method is that, when $p(x;\theta)$ is a multivariate Gaussian distribution, if the covariance matrices for different classes are assumed to be different, the optimal classifier provides a quadratic decision boundary. More interestingly, if the covariance matrices are assumed to be identical, then the optimal classifier has a linear decision boundary.

In general, the parametric approach works well when the number of features (i.e. dimension of x) is small. Unfortunately, the parametric approach suffers from a phenomenon known as "Curse of Dimensionality", which states that the number of required samples is an exponential function of the feature dimension. To address this problem, various regularization techniques [17] were proposed to obtain a robust estimate for situations with small sample size and/or high-dimension. Among them, the most famous one is the Regularized Discriminant Analysis [18] proposed by Friedman.

In many real world problems, data may form subgroups in each class. In this case, we may approximate the density $p(x;\theta)$ by a finite mixture model with,

$$\hat{p}(x;\theta)=\sum_{j=1}^{g}\pi_j p(x;\eta_j) \tag{2.5}$$

where g is the number of mixture components, π_j is a mixing proportion $\left(\pi_j > 0, \sum_{j=1}^{g}\pi_j = 1\right)$, and $p(x;\eta_j)$ is a component density function which depends on parameters η_j. There are three sets of parameters to be estimated: π_j, η_j and g. Generally, the value of g is the most difficult to estimate. Wolfe [19] proposed a modified likelihood ratio test to estimate the number of components. This approach was later investigated by Everitt [20] and Anderson [21]. Other approaches include Bozdogan [22] and Celeux and Soromenho [23]. Their approaches are based on the information-theoretic criteria such as entropy. If the value of g is already known, the values of π_j and η_j can be estimated by using the Expectation Maximization (EM) method [24], which is an extension of the maximum likelihood estimator for the uni-modal case (each class has only one group). Although the EM method is very easy to implement, its convergence rate can be poor, depending on the data distribution and the initial parameters. Jamshidian and Jennrich [25] proposed a generalized conjugate gradient method to speed up the EM method. Lindsay and Basak [26] described a method to initialize the EM model. In addition to EM, several authors also advocated the Markov Chain Monte-Carlo method [27], but this method is computationally demanding.

2.2.2 Nonparametric Methods

It is often the case in pattern recognition that assuming class conditional density functions to be members of a certain parametric family is not reasonable. The parameter estimation approach of the previous section is then not possible. Instead, we must estimate the class conditional density functions non-parametrically. In non-parametric estimation (or density estimation), we try to estimate $p(x|\Omega_i)$ in each point x whereas in parametric estimation we tried to estimate some unknown parameter vector. It is also possible to estimate directly the posterior probabilities $p(\Omega i|x)$ assuming that the training data was collected using mixed sampling. For non-parametric estimation we have two different well known methods: Parzen windows [28] and k-nearest neighbor's method [29, 30].

Both Parzen windows and k-NN methods are based on one simple idea - the class-conditional density $p(x|\Omega_j)$ may be approximated by the probability of training samples of class j falling into a small area around x divided by the volume of that area (for example, if we define the small area as the d-dimensional hypercube whose edges are h units long, the volume of that area is $V = h^d$). Specifically, the estimated class-conditional density $\hat{p}(x|\Omega_j)$ is,

$$\hat{p}(x|\Omega_j) = \frac{k_j/n_j}{V} \tag{2.6}$$

where V is the volume of the small area around x, k_j is the number of training samples of class j falling in the small area, and n_j is the number of training samples of class j. It can be shown that if we have infinite number of training samples of class j, as the volume of the small area approaches zero, the density estimate $\hat{p}(x|\Omega_j)$ in equation (2.6) asymptotically converges to the true density $p(x|\Omega_j)$.

Assume that the region D is a d-dimensional hypercube. If h_n is the length of the side of the hypercube, its volume is given by $V_n = h_n^d$. The window function that receives value 1 inside the hypercube centered at the origin, and value 0 outside the hypercube is defined as:

$$k(z) = \begin{cases} 1 & z_i \leq 0.5, i = 1, \ldots\ldots, d \\ 0 & otherwise \end{cases} \tag{2.7}$$

where $z \in R^d$. Then, $k\left(\frac{x - x_i}{h}\right)$ is 1, if x_i falls in the hypercube D centered at x, and 0 otherwise. Thus, the number of training samples of class j falling into the hypercube D is,

$$k_j = \sum_{i=1}^{n_j} k\left(\frac{x - x_i}{h}\right) \tag{2.8}$$

Substituting equation (2.8) into equation (2.6), we obtain the density estimate,

$$\hat{p}(x|\Omega_j) = \frac{1}{n_j V} \sum_{i=1}^{n_j} k\left(\frac{x - x_i}{h}\right) \tag{2.9}$$

It is to be noted that in contrast to the general density functions, because of the Parzen window function (2.7), this estimation is not continuous. To resolve this problem, continuous window functions are usually used.

There is more general approach to obtain the density estimation is available if some other k functions is considered than above. The Parzen-window density estimate at x using n training samples and the window function k is defined by

$$\hat{p}(x|\Omega_j) = \frac{1}{n_i}\sum_{i=1}^{n_j}\frac{1}{v_n}k\left(\frac{x-x_i}{h}\right) \tag{2.10}$$

The estimate $\hat{p}(x|\Omega_j)$ is an average of values of the window function at different points. Typically the window function has its maximum at the origin and its values become smaller when we move further away from the origin. Then each training sample is contributing to the estimate in accordance with its distance from x.

The important issue with the Parzen windows method is the selection of the window length h_n. This is not easy task, the choice often is application specific and it is rooted in the properties of the density estimate required by the application. If h_n is too large, then the density estimate $\hat{p}(x|\Omega_j)$ will be very smooth and 'out-of-focus'. If h_n is too small, then the estimate $\hat{p}(x|\Omega_j)$ will be just superposition of n sharp pulses centered at training samples, i.e. an erratic and noisy estimate of the true density, will not reflect the curvature of the true density. A general rule is that h should be small in high density areas and be large in low density areas.

Parzen estimates have some interesting properties, for $\hat{p}(x|\Omega_j)$ to be a legitimate density, it is required that

i. $\hat{p}(x|\Omega_j) \geq 0$ for all x and

ii. $\int \hat{p}(x|\Omega_j)dx = 1$,If we maintain the relation $h_n^d = V_n$.

With an unlimited number of training samples, it is possible to let V_n approach 0, and have $\hat{p}(x|\Omega_j)$ to converge to $p(x|\Omega_j)$. For this, it is required that:

i. The density function must be continuous.

ii. The window function must be bounded and a legitimate density.

iii. The values of the window function must be negligible at infinity.

iv. $V_n \rightarrow 0$ when $n \rightarrow \infty$

v. $nV_n \rightarrow \infty$ when $n \rightarrow \infty$

For pattern recognition, these optimality convergence properties are important, because with these properties we are able to show that the classification error of a classifier based on Parzen windows tends to the Bayes error when n approaches infinity. This holds also more generally: When the density estimates converge (in a certain sense) to the true class conditional density functions, and the prior probabilities are properly selected, then the error of the resulting classifier tends to the Bayes error.

In the Parzen windows method, the design of the classifier involved selecting window functions and suitable length of the window h_n . One possibility is let them depend on the training data. With respect to the general formulation of the density estimation, fixing

k_n and computing of suitable (small enough) V_n based on the selected k_n. To be more precise, center of the cell β_n into the test point x and let the cell grow until it encircles k_n training samples to estimate $p(x|\Omega_j)$. These k_n training samples are k_n nearest neighbors of x. Here, k_n is a given parameter. The k_n nearest neighbor (KNN) density estimate is given by

$$\hat{p}(x|\Omega_j) = \frac{k_n}{nV_n} \tag{2.12}$$

where V_n is the volume of the smallest possible x centered ball that contains k_n training samples, and n is the total number of training samples. If the true density is high in the proximity of x, the cell β_n is small and the density estimate is accurate. If the true density is low in the proximity of x, the cell β_n is large and the density estimate is not that accurate.

Let p(x) be continuous at x. The KNN density estimate converges if

i. $k_n \to \infty$ when $n \to \infty$

ii. $\frac{k_n}{n} \to 0$ when $n \to \infty$

One of the properties of the KNN density estimate is that although it is continuous, its partial derivatives are necessarily not continuous.

Two big issues with the k-NN method are the choice of k_n and the choice of the distance metric. Often in practice, k_n is chosen empirically and the Euclidean distance is used. But as it turns out, other distance measures, which incorporate prior knowledge, can significantly enhance the classification performance [31].

One of the biggest disadvantages for both the Parzen windows and the k-NN methods is that all training samples need to be retained in order to classify any given data. For large data sets, this leads to huge memory requirement and slow computation speed. To resolve this problem, several data reduction techniques, such as editing [32] and condensing [33], have been proposed. A branch and bound method [34] was also proposed to speed up the k-NN method. For the Parzen windows method, Silverman [35] proposed to use a Fourier transform to speed up the computation of kernel functions for univariate density estimation. In the multivariate case, procedures for approximating the kernel density using a reduced number of kernels were described by Fukunaga and Hayes [36] and Babich and Camps [37].

2.2.3 Linear Discriminant Functions and Classifiers

Parametric and non-parametric density estimation techniques find the decision boundaries by first estimating the probability distribution of the patterns belonging to each class. In the discriminant-based approach, the decision boundary is constructed explicitly. Knowledge of the form of the probability distribution is **not** required.

A discriminant function is a function of the pattern x that leads to a classification rule. For example, in a two-class problem, a discriminant function $h(x)$ is a function for which

$$\begin{aligned} h(x) &> k \Rightarrow x \in \Omega_1 \\ &< k \Rightarrow x \in \Omega 2 \end{aligned} \tag{2.11}$$

for constant *k*. In the case of equality $h(x) = k$, the pattern x may be assigned arbitrarily to one of the two classes. An optimal discriminant function for the two-class case is

$$h(x) = \frac{p(x/\Omega_1)}{p(x/\Omega_2)} \tag{2.12}$$

with $k = p(\Omega_1)/p(\Omega_2)$. Discriminant functions are not unique. If *f* is a monotonic function then

$$g(x) = f(h(x)) > k' \Rightarrow x \in \Omega_1$$

$$g(x) = f(h(x)) < k' \Rightarrow x \in \Omega_2$$

where $k' = f(k)$.

In the C group case we define C discriminant functions $g_i(x)$ such that

$$g_i(x) > g_j(x) \Rightarrow x \in \Omega_i \qquad j = 1 \ldots\ldots\ldots\ldots C \qquad j \neq i \tag{2.13}$$

That is, a pattern is assigned to the class with the largest discriminant, and for two classes, a single discriminant function

$$h(x) = g_1(x) - g_2(x) \tag{2.14}$$

with *k =0* reduces to the two-class case given by (1.11)

In linear discriminant analysis, for the two-class classification problem, we seek a weight vector $w \in R^d$ and a bias $b \in R$ such that,

$$w^T x + b \begin{cases} > 0 \\ < 0 \end{cases} \Rightarrow x \in \begin{cases} \Omega_1 \\ \Omega_2 \end{cases} \tag{2.15}$$

In the C-class case $(C > 2)$, a data point x will be assigned to the class k that satisfies,

$$g_k(x) = \max_j g_j(x) \tag{2.16}$$

where $g_j(x) = w_j^T x + b_j,\ w_j \in R^d \ and \ b_j \in R,\ j = 1,2,\ldots\ldots,c.$

Compared with the Parzen windows and k-NN methods, this approach has the advantages of small memory requirement and fast prediction speed, since only a few parameters (w_j, b_j) need to be used in prediction.

A common feature of various existing linear discriminant methods is that the parameters (the weights and the biases) are found by optimizing some criterion. An obvious criterion is to minimize the number of classification errors, which occur when using the training samples. Unfortunately, this criterion is difficult to handle because the number of errors is an integer. A simple extension is the perceptron criterion [38], which minimizes the total of the distances of the misclassified samples from the decision boundary. For the two-class classification problem, the perceptron criterion is,

$$\min J_p(w,b) = \sum_{x \in M} \left| w^T x + b \right| \tag{2.17}$$

where $w \in R^d$ is a weight vector, $b \in R$ is a bias, M is the set of misclassified samples, and $|\cdot|$ is the absolute value. The Perceptron has a very simple error-correction algorithm [39] for training. This training procedure cycles through the training samples, modifying the weight vector whenever a sample is misclassified, that is,

$$w(k+1) = w(k) + \eta x_i \tag{2.18}$$

where x_i is a training sample that has been misclassified by weights $w(k), \eta$ is the step size, and k is the iteration number. It can be proven that as long as the training samples are linearly separable (any pair of classes can be separated by a straight line), the error-correction algorithm converges to a solution in a finite number of iterations [39]. Nevertheless, even though the resulting straight line is able to perfectly separate training data, it cannot guarantee the quality of prediction on (additional) testing samples. Therefore, a separating line with a certain "margin" [40] relating to the boundary points was proposed to ease this problem (This "margin" idea was also used in developing support vector machines [41]). If the training samples are not linearly separable i.e. for hard problems, the error correction algorithm does not converge. Various methods [40] have been proposed to handle this problem, including "early stop", relaxation, linear programming approaches. The Perceptron can be generalized to handle the c-class classification problem by using "Kesler's Construction" [39].

The least squared error is another popular criterion. In this case, it is assumed that each sample x_i has a target value t_i. For example, $t_i = 1 \ \mathit{if} \ x_i \in \Omega_1$, $t_i = 0 \ \mathit{if} \ x_i \in \Omega_2$. The least squared error approach aims to minimize the total of the squared distances between the target values and the decision function outputs. For the two-class classification problem, the least squared error criterion is,

$$\min J_{lse}(w,b) = \sum_{i=1}^{n} \left\| t_i - \left(w^T x_i + b \right) \right\|^2 \qquad (2.19)$$

This criterion has a very nice property [3]. If we set the target value in such a way that $t_i = 1 \ \mathit{if} \ x_i \in \Omega_1$, $t_i = 0 \ \mathit{if} \ x_i \in \Omega_2$, as the number of training samples increases to infinity, the output value of the decision function, $g(x) = w^T x + b$, asymptotically converges to the difference between the posterior probabilities, $p(\Omega_1|x) - p(\Omega_2|x)$. Therefore, the approximate values of posterior probabilities can be computed from $p(\Omega_1|x) \approx (1+g(x))/2$ *and* $p(\Omega_2|x) \approx (1-g(x))/2$. Unlike the perceptrons, the least squared error rule does not necessarily produce a separable solution, even when the classes are linearly separable. Modifications of the least squared error rule have been proposed (e.g., the Ho-Kashyap procedure [42]) to remedy this problem, but the optimal approximation to the posterior probability is no longer achieved. Another interesting property of the least squared error rule is that with a proper choice of the target value t_i, the rule leads to the same result as Fisher's rule [43], for which the basic idea is to find the direction along which two classes are best separated. Furthermore, the least squared error rule can be easily extended to the multi-class case. The least squared error rule is the most popular criterion used for feed-forward neural networks.

For some difficult problems, a nonlinear decision boundary is required in order to obtain a good separation of different classes. One important generalization of the linear discriminant analysis is the generalized linear discriminant function. The basic idea is to nonlinearly map the data from the original d-dimensional space to some $\hat{d}$-dimensional space, then apply linear discrimination in the mapped space. For the c-class classification problem, the discriminant functions become,

$$g_j(x) = \sum_{i=1}^{\hat{d}} w_{ij}\phi_i(x) + b_j, \quad j = 1,2,\ldots\ldots,c \tag{2.20}$$

where $w_{ij} \in R$ is a weight, $b_j \in R$ is a bias, and $\phi_i(x): R^d \to R^{\hat{d}}$ is a nonlinear mapping that often called the "basis function". Hopefully, the transformed samples $\phi(x)$ are more linearly separable in the mapped space than the samples x were in the original space. Specht [44] and Flasinski and Lewichi [45] proposed the use of polynomial functions as the basis functions. But the problem with polynomial functions is that the number of terms increases rapidly with the order of the polynomial.

2.3 Structured Methods

Structural pattern recognition [46,47,48], sometimes referred to as syntactic pattern recognition due to its origins in formal language theory, relies on syntactic grammars to discriminate among data from different groups based upon the morphological interrelationships or interconnections present within the data. Structural pattern recognition techniques are effective for the data which contain an inherent, identifiable organization such as image data and time series data. The usefulness of structural pattern recognition systems, however, is limited as a consequence of fundamental complications associated with the implementation of the description and classification tasks. In many recognition problems involving complex patterns, it is more appropriate to adopt a hierarchical perspective where a pattern is viewed as being composed of simple sub patterns which are themselves built from yet simpler sub patterns . The elementary sub patterns to be recognized are called primitives and the given complex pattern is represented in terms of the interrelationships between these primitives. In syntactic pattern recognition, a formal analogy is drawn between the structure of patterns and the syntax of a language. The patterns are viewed as sentences belonging to a language, primitives are viewed as the alphabet of the language, and the sentences are generated according to a grammar. Thus, a large collection of complex patterns can be described by a small number of primitives and grammatical rules. The grammar for each pattern class must be inferred from the available training samples. Structural pattern recognition is intuitively appealing because, in addition to classification, this approach also provides a description of how the given pattern is constructed from the primitives. This paradigm has been used in situations where the patterns have a definite structure which can be captured in terms of a set of rules, textured images, and shape analysis of contours [48]. The implementation of a syntactic approach, however, leads to many difficulties which primarily have to do with the segmentation of noisy patterns and the inference of the grammar from training data. Fu [46] introduced the notion of attributed grammars which unifies syntactic and statistical pattern recognition. The syntactic approach may yield a combinatorial explosion of possibilities to be investigated, demanding large training sets and very large computational efforts [49].

Structural pattern recognition systems are difficult to apply to new domains because implementation of both the description and classification tasks requires domain knowledge. Knowledge acquisition techniques necessary to obtain domain knowledge from experts are tedious and often fail to produce a complete and accurate knowledge base. Consequently, applications of structural pattern recognition have been primarily restricted to domains in which the set of useful morphological features has been established in the literature and the syntactic grammars can be composed by hand. To overcome this limitation, a domain independent approach to structural pattern recognition is used that is capable of extracting morphological features and performing classification without relying on domain knowledge. A hybrid system that employs a statistical classification technique to perform discrimination based on structural features is a natural solution. While a statistical classifier is inherently domain independent, the domain knowledge necessary to support the description task can be eliminated with a set of generally useful morphological features. Such a set of morphological features is suggested as the foundation for the development of a suite of structure detectors to perform generalized feature extraction for structural pattern recognition in time series data.

Identification problems involving timeseries (or waveform) data constitute a subset of pattern recognition applications that is of particular interest because of the large number of domains that involve such data [50]. Both statistical and structural approaches can be used for pattern recognition of timeseries data: standard statistical techniques have been established for discriminant analysis of timeseries data, and structural techniques have been shown to be effective in a variety of domains involving timeseries data.

A domain independent structural pattern recognition system is one that is capable of acting as a "black box" to extract primitives and perform classification without the need for domain knowledge. Such a system would automatically describe and classify data, thereby eliminating the overhead associated with traditional approaches to structural pattern recognition. A domain independent structural pattern recognition system for timeseries data must incorporate techniques for the description and classification tasks that are not dependent on domain knowledge—i.e., generalized description and generalized classification. Since syntactic grammars are inherently tied to the domain and application, a sensible approach to generalized classification for timeseries data is a statistical classifier that performs discrimination based on structural features extracted from the data. Generalized description can be implemented using a foundation of generally useful morphological features that are effective regardless of the domain.

The field of signal processing offers a suggestion for morphological features that can provide the foundation for generalized description of timeseries data. Six fundamental types of modulation commonly used in signal processing systems—constant, straight, exponential, sinusoidal, triangular, and rectangular—entailmorphologies deliberately introduced into a continuous medium with the intent of conveying information regardless of the domain or application [51,52,53,54]. Moreover, these six modulation types subsume the small set of domain independent morphological features commonly extracted by structural pattern recognition systems—straight lines, parabolas, and peaks. A suite of feature extractors which identify morphological features based on these six modulation types, therefore, would constitute a first pass at implementing generalized feature extraction to support domain independent structural pattern recognition in timeseries data.

Structural approaches, while supported by psychological evidence which suggests that structure based description and classification parallels that of human perceptual and cognitive processes have not yet been developed to the fullest potential due to fundamental complications associated with implementing structural pattern recognition systems. Shiavi and Bourne [55] summarize the problems of applying structural methods for pattern recognition within the context of analyzing biological waveforms: There are obvious problems with the use of [structural techniques]. First, rather deep knowledge about the problem is required in order to successfully identify features and write [grammar] rules. While it is conceptually interesting to consider the possibility of using some automated type of grammatical inference to produce the rules, in practice no technique of grammatical inference has proved robust enough to be used with real problems involving biological waveforms. Hence, the writing of rules is incumbent on the designer of the analysis system. Similarly, the selection of features must be accomplished essentially by hand since automated techniques usually cannot provide the guidance necessary to make a useful feature selection. Second, the control strategy of typical parsing systems is relatively trivial and cannot deal with very difficult problems. Typical parsing techniques consist of simple repeated application of a list of rules, which is often equivalent to forward chaining, an elementary concept in knowledge based rule systems. Formation of a robust control strategy for guiding syntactic parsing of strings appears somewhat problematic. There is no general solution for extracting structural features from data. Friedman [56] addresses the issue by saying, "The selection of primitives by which the patterns of interest are going to be described depends upon the type of data and the associated application." Nadler [57] seems to support this position when he states, "...features are generally designed by hand, using the experience, intuition, and/or cleverness of the designer." The lack of a general approach for extracting primitives puts designers of structural pattern recognition systems in an awkward position: feature extractors are necessary to identify primitives in the data, and yet there is no established methodology for deciding which primitives to extract. The result is that feature extractors for structural pattern recognition systems are developed to extract either the simplest and most generic primitives possible, or the domain and application specific primitives that best support the subsequent classification task.

2.4 Neural Network Based Methods

The pattern recognition approaches discussed so far are based on direct computation through machines. Direct computations are based on statistical analysis techniques. The neural approach applies biological concepts to machines for pattern recognition. The outcome of this effort is invention of artificial neural networks. Neural networks can be viewed as massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections. Neural network models attempt to use some organizational principles (such as learning, generalization, adaptivity, fault tolerance, distributed representation, and computation) in a network of weighted directed graphs in which the nodes are artificial neurons and directed edges (with weights) are connections between neuron outputs and neuron inputs. The main characteristics of neural networks are that they have the ability to learn complex nonlinear input-output relationships, use sequential training procedures, and adapt themselves to the data. The most commonly used family of neural networks for pattern classification tasks [58] is the feed-forward network, which includes multilayer perceptron and Radial-Basis Function (RBF) networks. These networks are organized into layers and have unidirectional connections between the layers. Another popular network is the Self-Organizing Map

(SOM), or Kohonen-Network [59], which is mainly used for data clustering and feature mapping. The learning process involves updating network architecture and connection weights so that a network can efficiently perform a specific classification/clustering task. The increasing popularity of neural network models to solve pattern recognition problems has been primarily due to their seemingly low dependence on domain-specific knowledge (relative to model-based and rule-based approaches) and due to the availability of efficient learning algorithms. Neural networks provide a new suite of nonlinear algorithms for feature extraction (using hidden layers) and classification (e.g., multilayer perceptrons). In addition, existing feature extraction and classification algorithms can also be mapped on neural network architectures for efficient (hardware) implementation. In spite of the seemingly different underlying principles, most of the well known neural network models are implicitly equivalent or similar to classical statistical pattern recognition methods. Ripley [60] and Anderson et al. [61] also discuss the relationship between neural networks and statistical pattern recognition. Most neural networks conceal the statistics from the user. Despite these similarities, neural networks do offer several advantages such as, unified approaches for feature extraction & classification and flexible procedures for finding good, moderately nonlinear solutions. It consists of massive simple processing units with a high degree of interconnection between each unit. The processing units work cooperatively with each other and achieve massive parallel distributed processing. The design and function of neural networks simulate some functionality of biological brains and neurons systems. The advantages of neural networks are their adaptive-learning, self-organization and fault-tolerance capabilities.

For these outstanding capabilities, neural networks are used for pattern recognition applications. The goal in pattern recognition is to use a set of example solutions to some problem to infer an underlying regularity which can subsequently be used to solve new instances of the problem. Examples include hand-written digit recognition, medical image screening and fingerprint identification. In the case of feed-forward networks, the set of example solutions (called a training set), comprises sets of input values together with corresponding sets of desired output values. The training set is used to determine an error function in terms of the discrepancy between the predictions of the network, for given inputs, and the desired values of the outputs given by the training set. A common example of an error function would be the squared difference between desired and actual output, summed over all outputs and summed over all patterns in the training set. The learning process then involves adjusting the values of the parameters to minimize the value of the error function. This kind of feedback would be used to reconstruct the input patterns and make them free from error; thus increasing the performance of the neural networks. These kinds of networks are called as auto associative neural networks. As the name implies, they use back-propagation algorithms. However, effective learning algorithms were only known for the case of networks in which at most one of the layers comprised adaptive interconnections. Such networks were known variously as perceptrons [38] and Adalines[62], and were seriously limited in their capabilities [63]. Research into artificial neural network was stimulated during the 1980s by the development of new algorithms capable of training networks with more than one layer of adaptive parameters [64].

2.5 Fuzzy Based Methods

Pattern recognition tasks ideally suit fuzzy theory, as patterns are very often inexact, ambiguous, or corrupted. In the Handwritten Characters Recognition case example, the patterns to be recognized are not well defined and are ambiguous in many cases. Fuzzy rules for handwritten character recognition are given in Yamakawa [65]. Pattern recognition and classification are usually considered as very similar tasks. Classes can be described by fuzzy classification rules. Fuzzy rules can be used for classification purposes when the objects to be classified are noisy, corrupted, blurry, etc. Fuzzy rules can cope with those ambiguities. Contradictory fuzzy rules can be accommodated in one system, the tradeoff being achieved through the inference mechanism. This characteristic of fuzzy systems is very important for their applications in solving classification problems. Classification problems, when data examples labeled with class labels are available, have been successfully solved by classic statistical methods, especially when the data set is unambiguous and dense. If the data set is sparse in the problem state space, the problem is rather difficult. The confidence factors represent the percentage of instances of a given class which fall in a particular "patch" of the problem space:

Rule 1: IF A1 is M and A2 is S, THEN Class1 (CF = 1)

Rule 2: IF A1 is L and A2 is S, THEN Class1 (CF = 0.45)

Rule 3: IF A1 is L and A2 is S, THEN Class2 (CF = 0.55)

Rule 4: IF A1 is L and A2 is M, THEN Class2 (CF = 0.75)

Rule 5: IF A1 is L and A2 is M, THEN Class1 (CF = 0.25)

The above set of rules is ambiguous and contradictory. Rules 2 and 3 have the same antecedents, but different consequences, which is also true for rules 4 and 5. This situation is impossible for a symbolic production system to cope with. A fuzzy production system can infer a proper classification, as for every input data vector all rules may fire to some degree and all of them contribute to the final solution to different degrees. The points at the border between the two classes should be correctly classified by the fuzzy rules as they take support not only from one of the contradictory rules but from a rule that has lateral fuzzy labels with it.

When the data examples are not labeled with the class or pattern labels, that is, when the classes they belong to are not known, then different clustering techniques can be applied to find the groups, the clusters, in which data examples are grouped. The clusters show the typical patterns, the similarity, and the ambiguity in the data set. In addition to the exact clustering, fuzzy clustering can be applied too.

Fuzzy clustering is a procedure of clustering data into possibly overlapping clusters, such that each of the data examples may belong to each of the clusters to a certain degree. The procedure aims at finding the cluster centers V_i (i = 1, 2, . . .,c) and the cluster membership functions μ_i which define to what degree each of the n examples belong to the ith cluster. The number of clusters c is either defined a priori (supervised type of clustering) or chosen by the clustering procedure (unsupervised type of clustering). The result of a clustering procedure can be represented as a fuzzy relation $\mu_{i,k}$, such that:

$$\sum_{i=1}^{c} \mu_{1,k} = 1$$

for each k = 1, 2, . . .,n; (the total membership of an instance k to all the clusters equals 1)

$$\sum_{i=1}^{c} \mu_{1,k} > 0$$

for each i = 1, 2, . . .,c (there are no empty clusters) .

A widely used algorithm for fuzzy clustering is the C-means algorithm suggested by J. Bezdek [66,67]. Fuzzy clustering is an important data analysis technique. It helps to understand better the ambiguity in data. It can be used to direct the way other techniques for information processing are used afterward. For example, the structure of a neural network to be used for learning from a data set can be defined to a great extent after knowing the optimal number of fuzzy clusters.

2.6 Handwritten Character Recognition

For more than thirty years, researchers have been working on handwriting recognition. As in the case of speech processing, they aimed at designing systems able to understand personal encoding of natural language. This new stage in the evolution of handwriting processing results from a combination of several elements: improvements in recognition rates, the use of complex systems integrating several kinds of information, the choice of relevant application domains, and new technologies such as high quality high speed scanners and inexpensive powerful CPUs. Methods and recognition rates depend on the level of constraints on handwriting. The constraints are mainly characterized by the types of handwriting, the number of scriptors, the size of the vocabulary and the spatial layout. Obviously, recognition becomes more difficult when the constraints decrease. Considering the types of roman script (roughly classified as hand printed, discrete script and cursive script), the difficulty is lower for handwriting produced as a sequence of separate characters than for cursive script which has much in common with continuous speech recognition. For other writing systems, character recognition is hard to achieve, as in the case of Kanji which is characterized by complex shapes and a huge number of symbols.

The characteristics which constrain hand writting may be combined in order to define handwriting categories for which the results of automatic processing are satisfactory. The trade-off between constraints and error rates give rise to applications in several domains.

These new challenges bring the ongoing studies closer to unconstrained handwritten language processing which is the ultimate aim. The reading of all of the handwritten and printed information present on a document is necessary to process it automatically, to use content dependent criteria to store, access and transmit it and to check its content. Automatic handwritten language processing will also allow one to convert and to handle manuscripts produced over several centuries within a computer environment.

Recognition strategies heavily depends on the nature of the data to be recognized. In the cursive case, the problem is made complex by the fact that the writing is fundamentally ambiguous as the letters in the word are generally linked together, poorly written and may even be missing. On the contrary, hand printed word recognition is more related to printed word recognition, the individual letters composing the word being usually much easier to isolate and to identify. As a consequence of this, methods working on a letter basis (i.e., based on character segmentation and recognition) are well suited to hand printed word recognition while cursive scripts require more specific and/or sophisticated techniques. Inherent ambiguity must then be compensated by the use of contextual information.

Intense activity was devoted to the character recognition problem during the seventies and the eighties and pretty good results have been achieved [68]. Character Recognition techniques can be classified according to two criteria: the way preprocessing is performed on the data and the type of the decision algorithm. Preprocessing techniques include three main categories: the use of global transforms (correlation, Fourier descriptors, etc.), local comparison (local densities, intersections with straight lines, variable masks, characteristic loci, etc.) and geometrical or topological characteristics (strokes, loops, openings, diacritical marks, skeleton, etc.). Depending on the type of preprocessing stage, various kinds of decision methods have been used such as: various statistical methods, neural networks, structural matching (on trees, chains, etc.) and stochastic processing (Markov chains, etc.). Many recent methods mix several techniques together in order to provide a better reliability to compensate the great variability of handwriting.

Neural network computing has been expected to play a significant role in a computer-based system of recognizing handwritten characters. This is because a neural network can be trained quite readily to recognize several instances of a written letter or word, and can then be generalized to recognize other different instances of that same letter or word. This capability is vital to the realization of robust recognition of handwritten characters or scripts, since characters are rarely written twice in exactly the same form.

There have been reports of successful use of neural networks for the recognition of handwritten characters [69, 70], but we are not aware of any general investigation which might shed light on the systematic approach of a complete neural network system for the automatic recognition of cursive character.

The following is a brief description of several recognition algorithms for the handwritten characters.

Gader et al [71] reported a methodology using a pipeline strategy for handwritten numeral recognition which combines one eigenvalue filter, two stage template matchers (first stage has 754 templates and second stage has 657 templates) and 33 digit models. Correct rates of 94.03-96.39% and substitution error rates of 0.54- 1 .O5% were obtained over the real-world databases. However, this methodology is not ideal for real-time processing.

Le Cun et al [70] reported a neural network recognizer using a multilayer neural network. They use size normalized digit images as direct input to a neural network, with allowing the feature extraction and classification by multilayer neural network simultaneously. The neural network consists of four layer neurons such as 16x16 neurons in the input, 12x64 neurons in the first hidden layer, 12x16 neurons in the second hidden layer, 30 neurons in the third hidden layer and 10 output layer neurons. So, in total, there are 1256 neurons and 64,660 connections. They got an 86% rate of correct classification rate and a 13% rejection rate to get 1% substitution error rate.

Recently, with the rapid development of computer and parallel processing technologies, scientists found that recognition performance can be improved significantly by combining several methodologies. For instances, Suen et al [72] combined four algorithms for recognizing unconstrained handwritten numerals. They found that the combined algorithms complement each other in many ways to reduce the substitution error rate while maintaining a fairly high correct classification rate. Therefore, they got a correct rate of 93.05% and a rejection rate of 6.95% while maintaining a zero error rate. Cohen et al and Hull et al [73] also use four algorithms in parallel. The combined algorithms are a polynomial discriminant method, a method that relies on statistical and structural analysis on the contours of digits, a structural classifier and a contour analysis method.

Finally, we need to mention the current status of the recognition results. In the handwritten digit recognition, the best results have been reported with 85-95% correct classification rates and 1-5% substitution error rates [70,74,75,76].

2.7 Conclusion

The objective of this chapter was to introduce the various pattern recognition problems and its various existing solutions using neural network techniques and other well known techniques. Pattern recognition is a fast-moving and proliferating discipline. It is not easy to form a well-balanced and well-informed summary view of the newest developments in this field. It is still harder to have a vision of its future progress. In its early stage of development, statistical pattern recognition focused mainly on the core of the discipline: The Bayesian decision rule and its various derivatives (such as linear and quadratic discriminant functions), density estimation, the curse of dimensionality problem, and error estimation. Due to the limited computing power available in the 1960s and 1970s, statistical pattern recognition employed relatively simple techniques which were applied to small-scale problems. Since the early 1980s, statistical pattern recognition has experienced a rapid growth. Its frontiers have been expanding in many directions simultaneously. This rapid expansion is largely driven by the following forces.

i. Increasing interaction and collaboration among different disciplines, including neural networks, machine learning, statistics, mathematics, computer science, and biology. These multidisciplinary efforts have fostered new ideas, methodologies, and techniques which enrich the traditional statistical pattern recognition paradigm.

ii. The prevalence of fast processors, the Internet, large and inexpensive memory and storage. The advanced computer technology has made it possible to implement complex learning, searching and optimization algorithms which was not feasible a few decades ago. It also allows us to tackle large-scale real world pattern recognition problems which may involve millions of samples in high dimensional spaces (thousands of features).

iii. Emerging applications, such as data mining and document taxonomy creation and maintenance. These emerging applications have brought new challenges that foster a renewed interest in statistical pattern recognition research.

iv. Last, but not the least, the need for a principled, rather than ad hoc approach for successfully solving pattern recognition problems in a predictable way.

Structural pattern recognition can be a powerful analysis tool within domains where a description composed of morphological sub patterns and their interrelationships is paramount to accurate classification decisions. A structural pattern recognition system typically includes feature extractors to identify instances of morphological characteristics of the data which, in turn, are used as the basis for classification using syntactic grammars. The domain knowledge necessary to guide feature extractor and grammar development is gathered using knowledge acquisition techniques. However, such techniques are time consuming, inexact, and do not always produce a complete knowledge base of the domain. Consequently, structural approaches to pattern recognition are difficult to apply to unexplored or poorly understood domains, thus limiting them to domains where the feature types and the syntactic grammars have either become established in the literature or are obvious upon inspection of the data. Eliminating the effort necessary to implement feature extraction and classification for structural pattern recognition systems will widen the applicability of structural approaches to complex, poorly understood domains. This can be accomplished using domain independent techniques for feature extraction and classification. A domain independent structural pattern recognition system is one that is capable of extracting features and performing classification without the need for domain knowledge. Such a system can be implemented using a hybrid approach that incorporates structural features with statistical techniques for classification: the structural features retain the morphological information necessary for discrimination, while the statistical classifier avoids the need to develop syntactic grammars that are inherently domain and application specific. The solution to making feature extraction domain independent is to employ generalized feature extraction to identify instances of morphologies which have proven to be useful across domains. Structure detectors were implemented to approximate a timeseries data set with one of six morphologies—constant, straight, exponential, sinusoidal, triangular, and trapezoidal. A methodology for applying these structure detectors to a timeseries data set in a piecewise fashion was developed, producing either a homogeneous or heterogeneous sequence of structures that together best approximate the entire time series. The efficiency of these structure detectors to generate morphological features suitable for classification was assessed against three standard statistical techniques for feature extraction—the identity, Fourier, and wavelet transformations—using two databases having markedly different characteristics. The classification accuracies achieved when using the structure detectors were at least as good as (and often superior to) the classification accuracies achieved when using the statistical feature extractors. The ability of the structure detectors to generate morphological features that result in classification accuracies better than the baseline established by commonly used statistical techniques demonstrates that the morphologies identified by the suite of structure detectors constitute a useful set of structural feature types. Moreover, the classification accuracies achieved on the two disparate databases illustrate that the suite of structure detectors is capable of extracting features from data with various characteristics. Certainly it is possible to produce better classification accuracies with domain and application specific feature extractors developed with the assistance of a domain expert, but this is burdensome for well understood domains and impossible for domains that are poorly understood. What this suite of structure detectors offers is a starting point for extracting features which have been shown to be generally effective for classification, providing a springboard for domain exploration and the subsequent refinement of these structure detectors with the goal of producing a structural pattern recognition system targeted to a particular domain and application.

As discussed, the neural networks have the ability to learn complex nonlinear input-output relationships, use sequential training procedures, and adapt themselves to the data. Neural networks have also several advantages such as, unified approaches for feature extraction & classification and flexible procedures for finding good, moderately nonlinear solutions. The neural networks are now well established as an important technique for solving pattern recognition problems, and indeed there are already many commercial applications of feed-forward neural networks in routine use.

References:

[1] Source: http://www.acm.org/ubiquity/, Ubiquity, vol.5 (7), (2004).

[2] Watanabe, S. "Pattern Recognition: Human and Mechanical", New York: Wiley, (1985).

[3] Bishop, C.M., "Pattern Recognition and Machine Learning", Springer, ISBN 0-387-31073-8, (2006)

[4] Fu, K.S., "Recent Development in Pattern Recognition.", IEEE Transaction on Computer., vol. C-29(10),pp.845-854,(1980)

[5] Andrews, H.C., "Introduction to Mathematical Techniques in Pattern Recognition", Willy, (1972).

[6] Chen,C.H., "Statistical Pattern Recognition", Hayden Book Company,(1972).

[7] Fukunaga, K., "Introduction to Statistical Pattern Recognition", Academic Press,(1972).

[8] Tou, J.T., and Gonzalez, R.C., "Pattern Recognition Principles", Reading, Massachusetts: Addison-Wesley, (1974).

[9] Sklalansky, J., and Wassel, G.N., "Pattern Classifier and Trainable Machines", New York, Springer-Verlag, (1981).

[10] Hu, M.K., "Visual Pattern Recognition by Moment Invariants." ,IEEE Trans. Info. Theory, vol. 8(20), pp. 179-187, (1987).

[11] Zhen_Ping, L. and Bavarian, B., "Comparison of Neural Network and a Piece-wise Linear Classifier.", Pattern Recognition, letter, vol. 12, pp. 649-655, (1991).

[12] Kanal, L., "Patterns in Pattern Recognition:1968-1974.", IEEE Trans. Comp., vol. IT -20(6), pp. 697-722,(1974).

[13] Devijver, P.A. and Kittler, J., "Pattern Recognition: A Statistical Approach.", Englewood Cliffs, NJ, Prentice-Hall, (1982).

[14] Chen, C.H., "A Review of Statistical Pattern Recognition." , Pattern Recognition and Signal Processing, C.H. Chen(ed.), Sithoff and Noordhoff Publisher, pp. 117-132, (1978).

[15] Shimura, M., "Learning Procedures in Pattern Classification-introduction and Survey ",Proc.,4th International Joint Conference Pattern Recognition, Kyoto, Japan, (1978)

[16] Bayes, T., "An essay towards solving a problem in the doctrine of chances", Philosophical Transactions of the Royal Society (London), vol. 53, pp. 370-418, (1763).

[17] Mkhadri, A., Celeux G., and Nasroallah A., "Regularization in discriminant analysis: an overview", Computational Statistics and Data Analysis, vol. 23, pp. 403-423, (1997).

[18] Friedman,J.H., "Regularized discriminant analysis", Journal of the American Statistical Association, vol. 84, pp. 165-175, (1989).

[19] Wolfe, J.H. , "A Monte Carlo study of the sampling distribution of the likelihood ratio for mixtures of multinormal distributions.", Technical Bulletin STB 72-7, Navel Personnel and Training Research Laboratory, San Diego, CA, 92152, (1971).

[20] Everitt, B.S., "A Monte Carlo investigation of the likelihood ratio test for the number of components in a mixture of normal distributions", Multivariate Behavioral Research, vol. 16, pp. 171-180, (1981).

[21] Anderson, J.J., "Normal mixtures and the number of clusters problem", Computat. Statist. Quarterly, vol. 2, pp. 3-14, (1985).

[22] Bozdogan H., "Choosing the number of component clusters in the mixture-model using a new informational complexity criterion of the inverse-Fisher information matrix.", Proc. Third Conference of IFCS, Paris, (1992).

[23] Celex, G. and Soromenho G., "An entropy criterion for assessing the number of clusters in a mixture model.", Journal of Classification, vol. 13, pp. 195-212, (1996).

[24] Dempster, A.P., Laird, N.M. and Rubin, D.B., "Maximum-likelihood from incomplete data via the EM algorithm (with discussion)", Journal of the Royal Statistical Society, Series B, vol. 39, pp. 1-38, (1977).

[25] Jamshidian, M. and Jennrich, R. I., "Conjugate gradient acceleration of the EM algorithm", Journal of the American Statistical Association, vol. 88, pp. 221-228, (1993).

[26] Lindsay, B.G. and Basak, P., "Multivariate normal mixtures: a fast consistent method of moments", Journal of the American Statistical Association, vol. 88, pp. 468-476, (1993).

[27] Richardson, S. and Green,P., "On Bayesian analysis of mixtures with unknown number of components.", Journal of Royal Statistical Society, Series B, vol. 59, pp. 731-792, (1997).

[28] Parzen,E., "On estimation of a probability density function and mode.", Annals of Mathematical Statistics, vol. 33(2), pp. 1065-1076, (1962).

[29] Patrick, E.A. and Fischer,F.P., "A generalized k-nearest neighbor rule", Information and Control., vol. 16, No. 2, pp. 128-152, (1970).

[30] Fix, E. and Hodges, J.L., "Discriminatory analysis: nonparametric discrimination: consistency properties.", USAF School of Aviation Medicine, vol. 4, pp. 261-279, (1951).

[31] Simard, P., Cun, Y.L. and Denker,J., "Efficient pattern recognition using a new transformation distance", In Stephen J. Hanson, Jack D. Cowan, and C. Lee Giles, editors, Advances in Neural Information Processing Systems, vol. 5, pp. 50-58, Morgan Kaufmann, San Mateo, CA, (1993).

[32] Hand, D.J. and Batchelor, B.G., "An edited condensed nearest neighbor rule.", Information Sciences, vol. 14, pp. 171-180, (1978).

[33] Hart, P.E.,"The condensed nearest neighbor rule", IEEE Transactions on Information Theory, vol. 14, pp. 515-516, (1968).

[34] Fukunaga,K. and Narendra,P.M., "A branch and bound algorithm for computing k-nearest neighbors.", IEEE Transactions on Computers, vol. 24, pp. 750-753, (1975).

35] Silverman,B.W., "Kernel density estimation using the fast Fourier transform.", Applied Statistics, vol. 31, pp. 93-99, (1982).

36] Fukunaga,K., and Hayes,R.R., "The reduced Parzen classifier", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, No. 4, pp. 423-425, (1989).

37] Babich, G.A. and Camps,O.I., "Weighted Parzen windows for pattern classification", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18(5), 5, pp. 567-570, (1996).

[38] Rosenblatt,F., "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", Spartan Books, Washington, D.C., (1962).

[39] Nilsson,N.J., "Learning Machines : Foundations of Trainable Pattern Classifying Systems.", McGraw-Hill, New York, (1965).

[40] Duda, R.O. and Hart, P.E., "Pattern Classification and Scene Analysis, New York, Wiley, (1973).

[41] Boser,B.E., Guyon,I. and Vapnik,V., "A training algorithm for optimal margin classifiers", David Haussler, editor, Proceedings of the 4th Workshop on Computational Learning Theory(ACM Press, San Mateo, CA), pp. 144-152, (1992).

[42] Ho,Y.C. and Kashyap, R.L., "An algorithm for linear inequalities and its applications.", IEEE Transactions on Electronic Computers, vol. 14, pp. 683-688.

[43] Fisher, R.A., "The use of multiple measurements in taxonomic problems.", Annals of Eugenics, vol. 7(II), pp. 179-188, (1936).

[44] Specht,D.F., “Generation of polynomial discriminant functions for pattern recognition.”, IEEE Transactions on Electron. Comput., vol. 16, pp. 308-319, (1967).

[45] Flasinski,M. and Lewicki,G., “The convergent method of constructing polynomial discriminant functions for pattern recognition.”, Pattern Recognition, vol. 24(10), pp. 1009-1015, (1991).

[46] Fu, K.S., “Syntactic Pattern Recognition and Applications.” Prentice-Hall, Englewood Cliffs, New Jersey, (1982).

[47] Pavlidis, T., “Structural Pattern Recognition.”, SpringerVerlag, Berlin, (1977).

[48] Gonzalez, R. C. and Wintz P., “Digital Image Processing.”, Addison Wesley, (1987).

[49] Perlovsky,L.I., “Conundrum of Combinatorial Complexity.”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 20(6), pp. 666-670, (1998).

[50] Carpineto,C. and Romano,G., “A Lattice Conceptual Clustering System and its Application to Browsing Retrieval.”, Machine Learning, vol. 24(2), pp. 95-122, (1996).

[51] Antoniou, A., “Digital Filters: Analysis and Design.”, McGrawHill, New York, (1979).

[52] Bateman, A. and Yates, W., “Digital Signal Processing Design.”, Computer Science Press, New York, (1989).

[53] Coulon, F.D., “Signal Theory and Processing.”, Artech House, Dedham, Massachusetts, (1986).

[54] Mohanty, N., "Signal Processing: Signals, Filtering, and Detection.", Van Nostrand Reinhold, New York, (1987).

[55] Richard, G., Shiavi and Bourne., J.R., "Methods of Biological Signal Processing." Tzay Y. Young and King Sun Fu, editors, Handbook of Pattern Recognition and Image Processing (Academic Press, Orlando, Florida,)., pp. 545–568. (1986).

[56] Friedman, M. and Kandel, A. "Introduction to Pattern Recognition: Statistical, Structural, Neural, and Fuzzy Logic Approaches.", World Scientific, Singapore, (1999).

[57] Nadler, M. and Smith, E.P., "Pattern Recognition Engineering.", Wiley, New York, (1993).

[58] Jain, A.K., Mao, J. and Mohiuddin, K.M., "Artificial Neural Networks: A Tutorial.", Computer, pp. 31-44, (1996).

[59] Kohonen,T., "Self-Organizing Maps.", Springer Series in Information Sciences, vol. 30, (1995).

[60] Ripley, B., "Statistical Aspects of Neural Networks.", Networks on Chaos: Statistical and Probabilistic Aspects. U. Bornndorff-Nielsen, J. Jensen, and W. Kendal, eds., Chapman and Hall, (1993).

[61] Anderson, J., Pellionisz, A. and Rosenfeld, E., "Neurocomputing 2: Directions for Research.", Cambridge Mass.: MIT Press, (1990).

[62] Widrow,B. and Lehr,M.A.,"30 years of adaptive neural networks: perceptron, madeline, and backpropagation.", Proceedings of the IEEE 78 (9), pp. 1415-1442. (1990).

[63] Minsky, M. L. and Papert, S.A., “Perceptrons.”, Cambridge, MA: MIT Press. Expanded Edition,(1990).

[64] Rumelhart, D.E., Hinton, G.E. and Williams,R.J., “Learning internal representations by error propagation.”,Parallel Distributed Processing: Explorations in the Microstructure of Cognition Cambridge, MA: MIT Press. Reprinted in Anderson and Rosenfeld, vol. 1, pp. 318-362. (1988).

[65] Yamakawa, T.,”Pattern recognition hardware system employing a fuzzy neuron.”, Proceedings of the International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, pp 943-948,(1990).

[66] Bezdek, J., “Analysis of Fuzzy Information.”, Boca Raton, Fla, CRC Press,(1987).

[67] Bezdek, J. and Pal, S., “Fuzzy Models for Pattern Recognition.”, New York, IEEE Press,(1992).

[68] Mori, S., Suen, C.Y. and Yamamoto,K., “Historical review of OCR research and development.” ,Proceedings of the IEEE (Special Issue on Optical Character Recognition), vol. 80(7), pp. 1029--1058, (1992).

[69] Fukushima, K and Wake, N., "Handwritten alphanumeric character recognition by the neocognitron.", IEEE Trans. on Neural Networks, vol. 2(3), pp. 355-365, (1991).

[70] Cun, Y.L., Boser, B., Denker, J. S., Henderson, D., Howard, R.E., Hubbad, W. and Jackel,L.D., "Handwritten digit recognition with a backpropagation network", Neural Information Processing Systems, Touretzky editor, Morgan Kaufmann Publishers, vol. 2, pp. 396-404, , (1990).

[71] Gader, P., Forester, B., Ganzberger, M., Gillies, A., Mitchell, B., Whalen, M. and Yocum, T., "Recognition of handwritten digits using template and model matching.", Pattern Recognition, vol. 24(5), pp. 421-431, (1991).

[72] Suen, C. Y., Nadal, C., Legault, R.. Mai, T. A. and Lam, L., "Computer Recognition of unconstrained handwritten numerals.", Proc. of IEEE, vol. 80(7), pp 1162-1180, (1992).

[73] Cohen, E., Hull, J. J. and Srihari, S. N., "Understanding handwritten text in structural environment: Determining Zip codes from addresses." , International Journals of Pattern Recognition and Artificial Intelligence, vol. 5. pp. 221-264, (1991).

[74] Nadal, C. and Suen, C. Y., "Applying human knowledge to improve machine recognition of confusing handwritten numerals", Pattern Recognition, vol. 26(3), pp. 381-389, (1993).

[75] Cun,Y.L., Jackel,L.D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R E. and Hubbard, W., "Handwritten digit recognition: applications of neural network chips and automatic learning.", IEEE Communications Magazine, pp. 41- 46, (November 1989).

[76] Dreyhs, D., "Neural networks for the automatic recognition of handwritten digits", Applications of Neural Networks, Schuster editor, pp. 34-60, (1992).

CHAPTER 3

Soft Computing: Techniques, Models & Applications

Chapter 3: Soft Computing: Techniques, Models and Applications

Abstract

Soft computing is an innovative approach for constructing computationally intelligent systems, which can exhibit the improved performance for the pattern recognition task. It is now realized that complex real world problems require intelligent system that combine knowledge, techniques and methodologies from various sources. These intelligent systems are supported to posses the human like expertise within a specific domain, adopt themselves and learn to do better in changing environments and explain how they make decision and take actions. In confronting the real world computing problems, it is frequently advantageous to use several computing technique synergistically rather than exclusively, resulting in construction of complementary hybrid intelligent systems. In this chapter, we are discussing the various models and techniques of soft computing for constructing the hybrid systems.

3.1 Introduction

Soft computing refers to a collection of computational techniques in computer science, artificial intelligence, machine learning and some engineering disciplines, which attempt to study, model, and analyze very complex phenomena: those for which more conventional methods have not yielded low cost, analytic, and complete solutions. Earlier computational approaches could model and precisely analyze only relatively simple systems. More complex systems arising in biology, medicine, the humanities, management sciences, and similar fields often remained intractable to conventional mathematical and analytical methods. It should be pointed out that simplicity and complexity of systems are relative, and many conventional mathematical models have been both challenging and very productive.

Soft computing, a concept introduced by L.A. Zadeh in the early 1990s, is an evolving collection of methodologies for the representation of the ambiguity in human thinking. The core methodologies of soft computing are fuzzy logic, neural networks, and evolutionary computation. Soft computing targets at exploiting the tolerance for imprecision and uncertainty, approximate reasoning; optimize solutions and partial truth in order to achieve tractability, robustness, and low-cost solutions.

The study of neural networks has been started dates back to the 1940s. The neural networks are a well-established computational paradigm in the field of artificial intelligence (AI) [1 – 4]. The powerful penetration of neural networks is due to their strong learning and generalization capability. They are usually seen as a method for implementing complex nonlinear mappings (functions) using simple elementary units that are connected together with weighted, adaptable connectionist. We concentrate on optimizing the connection structure of the networks.

The theory of fuzzy logic and fuzzy sets was introduced by L.A. Zadeh in 1965. Fuzzy logic provides a means for treating uncertainty and computing with words. This is especially useful to mimic human recognition, which skillfully copes with uncertainty. Fuzzy systems are conventionally created from explicit knowledge expressed in the form of fuzzy rules, which are designed based on experts' experience. A fuzzy system can explain its action by fuzzy rules. Fuzzy systems can also be used for function approximation. The synergy of fuzzy logic and neural networks generates neurofuzzy systems, which inherit the learning capability of neural networks and the knowledge-representation capability of fuzzy systems.

Evolutionary algorithms (EAs) [5-15], inspired by the principles of biological evolution, are another paradigm in AI that has received much attention lately. Evolutionary computation is a computational method for obtaining the best possible solutions in a huge solution space based on Darwin's survival of the fittest principle. Evolutionary algorithms are a class of robust adaptation and global optimization techniques for many hard problems. Among evolutionary algorithms, the genetic algorithm is the best known and most studied, while evolutionary strategy is more efficient for numerical optimization. More and more biologically or nature-inspired algorithms are emerging. Evolutionary computation has been applied for the optimization of the structure or parameters of neural networks, fuzzy systems, and neurofuzzy systems. The hybridization between neural network, fuzzy logic, and evolutionary computation provides a powerful means for solving the real world problems of computation and predictions.

3.2 The Neural Networks

The study of the neural networks is generated by the study of human brain, that how the brain works to make the decision, to store the patterns and how a brain learns for the recognition of objects? The study defines that a human brain contains an average of 3×10^{10} neurons [16] of various types, which are basically known as decision elements, with each neuron connecting to up to 10^4 synapses [17]. Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain [18-20]. Neural networks are attractive since they consist of many neurons, each of the neurons processes information separately and simultaneously. All the neurons are connected by synapses with variable weights. Thus, neural networks are actually parallel distributed processing systems.

During the decades of 1940s McCulloch and Pitts found that the neuron can be modeled as a simple threshold device to perform logic function [21]. After that during the late 1940s, Hebb proposed the Hebbian rule [22] to describe how learning affects the synaptics between two neurons. In the late 1950s and early 1960s, Rosenblatt [23] proposed the perceptron model, and Widrow and Hoff [24] proposed the adaline (adaptive linear element) model, trained with a least mean squares (LMS) method. The various studies and development of neural networks have been proposed by the researchers [25-28]. The landmark of the field is the multilayer perceptron (MLP) model trained with the backpropagation (BP) learning algorithm published in 1986 by Rumelhart *et al.* [29]. Later it turned out that the BP algorithm had already been described in 1974 by Werbos [30] when he studied social problems.

3.2.1 Neurons

The human body is made up of a vast array of living cells. Certain cells are interconnected in a way that allows them to communicate pain, or to actuate fibers or tissues. Some cells control the opening and shutting of minuscule valves in the veins and arteries. Others tell the brain that they are experiencing cold, heat, or any number of sensations. These specialized communication cells are called neurons. Figure3.1 shows the biological structure of neurons.

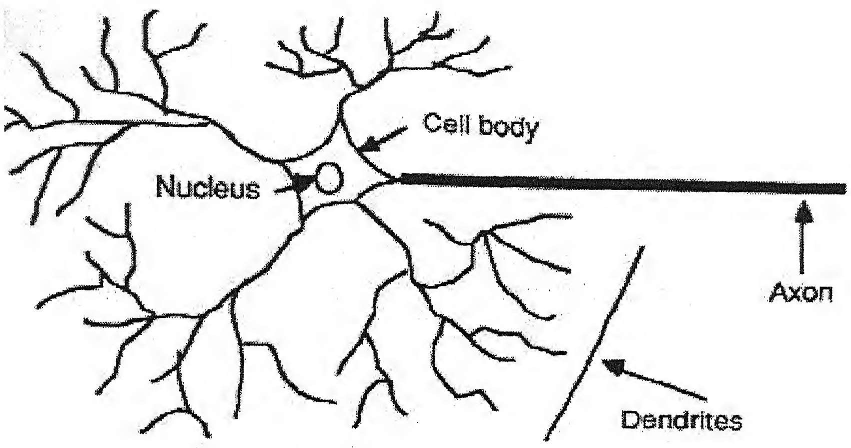


Figure 3.1: A biological neuron.

The biological neurons are equipped with long tentacle-like structures that stretch out from the cell body, permitting them to communicate with other neurons. The tentacles that take in signals from other cells and the environment itself are called dendrites, while the tentacles that carry signals from the neuron to other cells are called axons. The interaction of the cell body itself with the outside environment through its dendritic connections and the local conditions in the neuron itself cause the neuron to pump either sodium or potassium in and out, raising and lowering the neuron's electrical potential. When the neuron's electrical potential exceeds a threshold, the neuron fires, creating an action potential that flows down the axons to the synapses and other neurons. The action potential is created when the voltage across the cell membrane of the neuron becomes too large and the cell "fires," creating a spike that travels down the axon to other neurons and cells. If the stimulus causing the buildup in voltage is low, then it takes a long time to cause the neuron to fire. If it is high, the neurons fire much faster. The neuron is the basic information processing unit of a NN consists of:

i. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$

ii. An adder function, for computing the weighted sum of the inputs.

iii. Activation function (squashing function) for limiting the amplitude of the neuron output. The activation function represents a linear or nonlinear mapping from the input to the output.

The output of the neuron is calculated by

$$u = \sum_{i=1}^{n} w_i x_i - \theta \tag{3.1}$$

and the final output z is calculated as:

$$z = f(u) \tag{3.2}$$

There are some functions that can be used as activation functions as follows:

1. Hard limiter

$$f(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u \prec 0 \end{cases} \tag{3.3}$$

2. Piecewise linear function

$$f(u)=\begin{cases}1, & u \succ \frac{1}{2} \\ u & -\frac{1}{2} \le u \le \frac{1}{2} \\ 0 & u \prec -\frac{1}{2}\end{cases} \tag{3.4}$$

3. Logistic Sigmoid function

$$f(u)=\frac{1}{1+e^{-\beta u}} \tag{3.5}$$

4. Hyperbolic tangent function

$$f(u)=\tanh(\beta u) \tag{3.6}$$

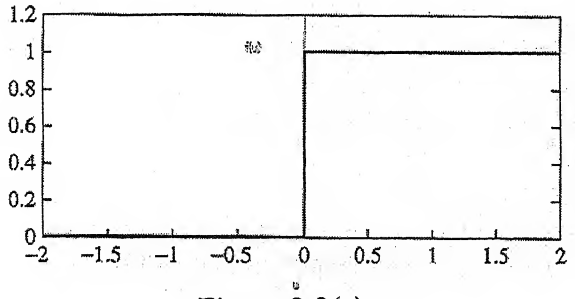


Figure 3.2(a)

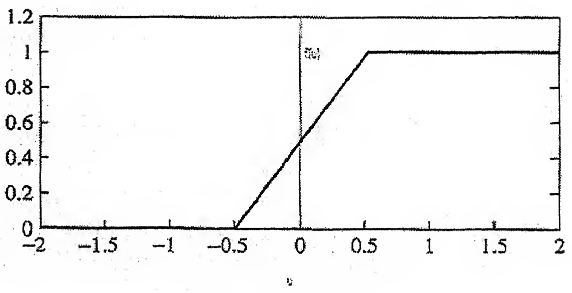


Figure 3.2(b)

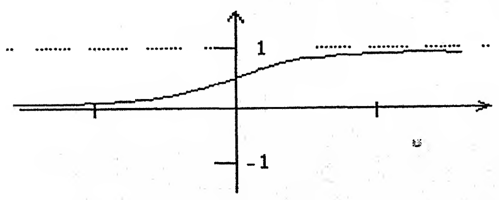


Figure 3.2(c)

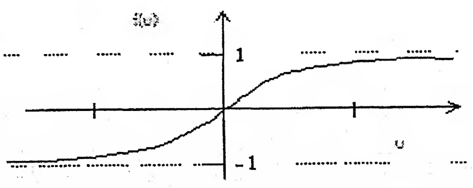


Figure 3.2(d)

Figure 3.3: Different type of activation functions, (a) Hard-limiter (b) Piecewise linear(c) Sigmoid (d) Hyperbolic tangent.

All the above functions are monotonically increasing with the domain of output $(-1,\ 1)$ or $(0,\ 1)$. In practical implementations, all the neurons are typically assumed to have the same activation functions. Depending on the type of processor, the calculation of nonlinear activation functions may consume considerable time [31].

The most commonly used transfer function is the sigmoid or logistic function, because it has nice mathematical properties such as monotonicity, continuity, and differentiability, which are very important when training a neural network with gradient descent. Initially, scientists studied the single neuron with a hard-limiter or step-transfer function. McCulloch and Pitts used an "all or nothing" process to describe neuron activity [21]. Rosenblatt [23] used a hard limiter as the transfer function and termed the hard-limiter neuron a perceptron because it could be taught to solve simple problems. The hard-limiter is an example of a linear equation solver with a simple line forming the decision boundary.

The most common activation function is the logistic sigmoid function, which is given by the following equation:

$$z(output) = \frac{1}{1+e^{-u}} \tag{3.7}$$

The value of u is replaced from the equation 3.1, we get:

$$z(output) = \frac{1}{1+e^{-\left(\sum_{i=1}^{n} w_i + \theta\right)}} \tag{3.8}$$

3.2.2 The Neural Network Architecture

The connection weight matrix **W** = [w_{ij}], where w_{ij} denotes the connection weight from node *i* to node *j*, is used to describe the network architecture. When w_{ij} = 0, there is no connection from node *i* to node *j*. By setting the connection weights between nodes as zero, one can realize different network topologies. Basically, all artificial neural networks have a similar structure or topology as shown in figure 3.3. In that, structure some of the neurons interfaces to the real world to receive its inputs. Other neurons provide the real world with the network's outputs. This output might be the particular character that the network thinks that it has scanned or the particular image it thinks is being viewed. All the rest of the neurons are hidden from view.

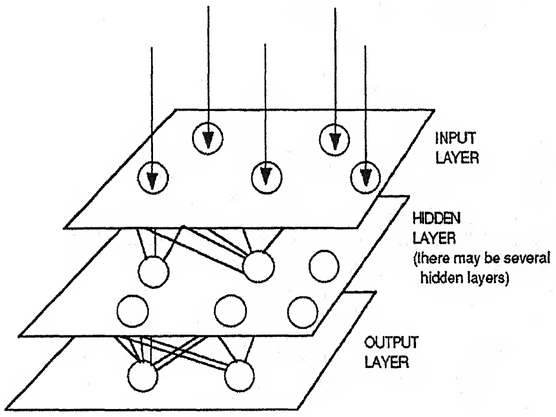


Figure 3.3: A simple neural network diagram

According to the architecture, neural networks can be grossly classified into feedforward neural networks (FNNs), recurrent neural networks (RNNs), and their combinations. Some popular network topologies include fully connected layered FNNs, RNNs, lattice networks, layered FNNs with lateral connections. The nonzero elements of **W** can be adapted by a learning algorithm. In an FNN, the connections between neurons are in a feedforward manner. The network is usually arranged in the form of layers. In layered FNNs, there is no connection between the neurons within each layer, and no feedback between layers. A fully connected layered FNN is a network such that every node in any layer is connected to every node in its adjacent forward layer. When some of the connections are missing, it becomes a partially connected layered FNN. FNNs exhibit no dynamic properties and the networks are simply a nonlinear mapping. The popular MLP and RBFN are fully connected layered FNNs. In an RNN, there is at least one feedback connection that corresponds to an integration operation or unit delay. Thus, an RNN actually represents a nonlinear dynamic system.

Most applications require networks that contain at least the three normal types of layers - input, hidden, and output. The layer of input neurons receives the data either from input files or directly from electronic sensors in real-time applications. The output layer sends information directly to the outside world, to a secondary computer process, or to other devices such as a mechanical control system. Between these two layers there can be many hidden layers. These internal layers contain many of the neurons in various interconnected structures. The inputs and outputs of each of these hidden neurons simply go to other neurons. In most networks, each neuron in a hidden layer receives the signals from all of the neurons in a layer above it, typically an input layer. After a neuron performs its function it passes its output to all of the neurons in the layer below it, providing a feed forward path to the output.The way that the neurons are connected to each other has a significant impact on the operation of the network.

3.2.3 Training of ANN

After finalizing the architecture of the neural network for a given application, the training or learning is required for getting the desired output from the network. Training or learning of a neural network is an optimization process that produces an output that is as close as possible to the desired output by adjusting network parameters. This kind of parameter estimation is also called learning or *training algorithm*. Neural networks are usually trained by epoch. An epoch is a complete run when all the training examples are presented to the network and are processed using the learning algorithm only once. After learning, a neural network represents a complex relationship, and possesses the ability for generalization. When a new input is presented to the trained neural network, a reasonable output is produced. Learning methods are conventionally divided into supervised, unsupervised, reinforcement, and evolutionary learning. Supervised learning is widely used in pattern recognition, approximation, control, modeling and identification, signal processing, and optimization. Reinforcement learning is usually used in control. Unsupervised learning schemes are mainly used for pattern recognition, clustering, vector quantization, signal coding, and data analysis. Evolutionary computation is a class of optimization techniques, which can be used to search for the global minima/maxima of an objective function. Evolutionary learning is used for adjusting neural network architecture and parameters using an evolutionary algorithm (EA), and can also be used to optimize the control parameters in a supervised or unsupervised learning algorithm.

Supervised learning is based on a direct comparison between the actual network output and the desired output. Network parameters (weights) are adjusted by a combination of the training pattern set and the corresponding errors between the desired output and the actual network response. The errors first calculated then propagated back through the system, causing the system to adjust the weights, which evolve the learning process. The pattern set, which enables the learning, is called the "training set." During the learning of a network the same set of data is processed many times as the connection weights are ever refined. So supervised learning can be defined as a closed-loop feedback system, where the error is the feedback signal. The trained network is used to emulate the system. To control a learning process, a criterion is needed to decide the time for terminating the process. For supervised learning, an error measure, which shows the difference between the network output and the output from the training samples, is used to guide the learning process. The error measure is usually defined by the mean squared error and calculated by the error function:

$$E = \frac{1}{N}\sum_{p=1}^{N}\left|z_p - \hat{z}_p\right|^2 \tag{3.9}$$

Where the N is the total number of patterns pair from a sample training set, z_p is the actual output and $\hat{z}_p$ is the output calculated by the network for p^{th} pair of sample of training set. This function is also known as the objective function to optimize the network. The error E is calculated a new after each epoch. This process of network training is terminated when E is sufficiently small or a failure criterion is met. To minimize the error up to the non significant value, a gradient-descent procedure is usually applied. The LMS [24] and back propagation algorithms [29] are two early, but most popular, supervised learning algorithms. Both of them are derived using a gradient-descent procedure. When finally, the system has been correctly learned, and no further learning is needed, the weights can, if desired, be "frozen." In some systems, this finalized network is then turned into hardware so that it can be fast. Other systems don't lock themselves in but continue to learn while in production use.

Unsupervised learning involves no target values. It tries to auto associate information from the inputs to decide what features it will use to group the input data. Unsupervised learning is solely based on the correlations among the input data, and is used to find the significant patterns or features in the input data without any supervision. A criterion is needed to terminate the learning process. Without a termination criterion, a continuous learning process continues even when a pattern, which does not belong to the training patterns set, is presented to the network. The network is adapted according to a constantly changing environment. Hebbian learning [22], competitive learning [26], and Kohonen's SOM [32,33] are the three mostly used unsupervised learning approaches. In general the unsupervised learning is slow to settle into stable conditions. In Hebbian learning [22], learning is a purely local phenomenon, involving only two neurons and a synapse. The synaptic weight change is proportional to the correlation between the pre and postsynaptic signals. The *C*-means algorithm is a popular competitive learning-based clustering method [34]. By using the correlation of the input vectors, the learning rule changes the network weights to group the input vectors into clusters. The Boltzmann machine [35] uses a kind of stochastic training technique known as SA [36], which can been treated as a special type of unsupervised learning based on the inherent property of a physical system. Tuevo Kohonen, an electrical engineer at the Helsinki University of Technology developed a self-organizing network [37], sometimes called an autoassociator, that learns without the benefit of knowing the right answer. It is an unusual looking network in that it contains one single layer with many connections. The weights for those connections have to be initialized and the inputs have to be normalized.

The neurons are set up to compete in a winner-take-all fashion. The other most common algorithm of unsupervised learning is the Hopfield neural network model [38,39] of associative memory. The Hopfield neural network suggested by Hopfield, in which he used energy estimation function and relates the network to other physical systems. Hopfield network is fully interconnected network with symmetric weights, no self-feedback and asynchronous updation of the state of processing elements.

Reinforcement learning [40] is a special case of supervised learning, where the exact desired output is unknown. It is based only on the information as to whether or not the actual output is close to the estimate. Explicit computation of derivatives is not required. This, however, presents a slower learning process. Reinforcement learning is a learning procedure that *rewards* the neural network for its *good* output result and *punishes* it for the *bad* output result. It is used in the case when the correct output for an input pattern is not available and there is need for developing a certain output. The evaluation of an output as *good* or *bad* depends on the specific problem and the environment. For a control system, if the controller still works properly after an input, the output is judged as *good*; otherwise, it is considered as *bad*. The evaluation of the output is binary, and is called *external reinforcement*. Thus, reinforcement learning is a kind of supervised learning with the external reinforcement as the error signal. Reinforcement learning can learn the system structure by trial-and-error, and is suitable for online learning [41-44].

Evolutionary learning approach is attractive since it can handle the global search problem better on a vast, complex, multimodal, and no differentiable surface. It is not dependent on the gradient information of the error (or fitness) function, and thus is particularly appealing when this information is unavailable or very costly to obtain or estimate. Evolutionary Algorithms can be used to search for the optimal control parameters in supervised as well as unsupervised learning by optimizing their respective objective functions. It can also be used as an independent training method for network parameters by optimizing the error function. Evolutionary Algorithms are widely used for training neural networks and tuning fuzzy systems, and are generally much less sensitive to the initial conditions. They always search for a globally optimal solution, while supervised and unsupervised learning algorithms can only find a local optimum in a neighborhood of the initial solution [45].

3.2.4 Characteristics of ANN

The overall functioning of neural networks is divided into two stages: learning (training) and generalization (recalling). Network training is done by giving the input samples, and network parameters are adapted using a learning method. This can be done in an online or offline manner. Once the network is trained to accomplish the desired performance, the learning process is terminated. For real-time applications, a neural network is required to have a constant processing delay regardless of the number of input nodes, and a minimum number of layers. As the number of input nodes increases, the size of the network layers should grow at the same rate without additional layers. The Artificial neural networks are characterized by the network architecture, node characteristics, and learning rules. Neural networks are usually biologically motivated. Each neuron is a computational node, which represents a nonlinear function. Neural networks possess the following advantages:

Neural networks have strong learning capability. They can adapt themselves by changing the network parameters in a surrounding environment. Powerful learning algorithms make this capability possible. Learning and generalization are the most salient features of neural networks.

A well-trained neural network has superior generalization capability. This can be attributed to the bounded and smooth nature of the hidden-unit responses. The bounded-unit response localizes the nonlinear effects of the individual hidden units in a neural network and allows for the approximations in different regions of the input space to be independently tuned [42]. In contrast, in conventional curve-fitting methods, the polynomials and other functions have a potential divergence nature.

Some neural networks such as the SOM [44] and competitive learning based neural networks have a self-organization property. The training of these networks is based on the unsupervised learning algorithms.

Neural networks have robustness and fault-tolerant capability. A neural network can easily handle imprecise, fuzzy, noisy, and probabilistic information. It is a distributed information system, where information is stored in the whole network in a distributed manner by the network structure. Thus, the overall performance does not degrade significantly when the information at some node is lost or some connections in the network are damaged. The network will immediately improve the performance by updating the connection weights using the learning rule and the current result. Therefore, the network can repair itself, and possesses a strong fault-tolerant capability. Some neural networks such as the Hopfield network can be used as associative storage of information.

When a noisy or incomplete pattern is presented to a trained network, it will help to find the correct pattern; that is, the trained network is fault tolerant.

The massive parallelism using simple uniform units can be readily implemented in analog VLSI or optical hardware [45,46], or be implemented on special-purpose massively parallel hardware [47].

3.3 The Backpropagation Learning Algorithm

The backpropagation (BP) learning algorithm is currently the most popular supervised learning rule for performing pattern classification tasks [24,29,30]. It is not only used to train feed forward neural networks such as the multilayer perceptran, it has also been adapted to recurring neural networks [48]. The BP algorithm is a generalization of the delta rule, known as the least mean square *algorithm* [24]. Thus, it is also called the *generalized delta rule.* The BP overcomes the limitations of the perceptron learning enumerated by Minsky and Papert [49].Due to the BP algorithm, the MLP can be extended to many layers. The BP algorithm propagates backward the error between the desired signal and the network output through the network. After providing an input pattern, the output of the network is then compared with a given target pattern and the error of each output unit calculated. This error signal is propagated backward, and a closed-loop control system is thus established. The weights can be adjusted by a gradient-descent-based algorithm. In order to implement the BP algorithm, a continuous, nonlinear, monotonically increasing, differentiable activation function is required. The two most-used activation functions are the logistic function (3.5) and the hyperbolic tangent function (3.6), and both are sigmoid functions.

We want to train a multi-layer feed forward network by gradient descent to approximate an unknown function, based on some training data consisting of pairs $(x\ ,z\ ) \in S$. The vector $\boldsymbol{x}$ represents a pattern of input to the network, and the vector z the corresponding desired output from the training set $\boldsymbol{S}$. The objective function for optimization is defined as the error MSE can be calculated by equation (3.9).

All the network parameters $W^{(m-1)}$ and θ^m, $m = 2 \cdot \cdot \cdot M$, can be combined and represented by the matrix $W = \lfloor w_{ij} \rfloor$. The error function E can be minimized by applying the gradient-descent procedure as:

$$\Delta\ W = -\eta \frac{\partial E}{\partial W} \tag{3.10}$$

where η is a learning rate or step size, provided that it is a sufficiently small positive number.

Applying the chain rule the equation (3.10) can express as

$$\frac{\partial E}{\partial w_{ij}^{(m)}} = \frac{\partial E}{\partial u_j^{(m+1)}} \frac{\partial u_j^{(m+1)}}{\partial w_{ij}^{(m)}} \tag{3.11}$$

while $$\frac{\partial u_j^{(m+1)}}{\partial w_{ij}^{(m)}} = \frac{\partial}{\partial w_{ij}^{(m)}} \left(\sum w_j^{(m)} o^{(m)} + \theta_j^{(m+1)} \right) = o_i^{(m)} \tag{3.12}$$

and $$\frac{\partial E}{\partial u_j^{(m+1)}} = \frac{\partial E}{\partial o_j^{(m+1)}} \frac{\partial o_j^{(m+1)}}{\partial u_j^{(m+i)}} = \frac{\partial E}{\partial o_j^{(m+1)}} \phi_j^{(m+1)} \left(u_j^{(m+1)} \right) \tag{3.13}$$

For the output unit m=M-1

$$\frac{\partial E}{\partial o_j^{(m+1)}} = e_j \tag{3.14}$$

For the hidden units, $m = 1, 2, 3 \ldots\ldots\ldots, M-2$,

$$\frac{\partial E}{\partial o_j^{(m+1)}} = \sum_{\omega=2}^{j_{m+2}} \frac{\partial E}{\partial u_{\omega}^{m+2}} \omega_{j\omega}^{m+1} \tag{3.15}$$

Define the delta function by

$$\delta_j^{(m)} = \frac{\partial E}{\partial u_p^{(m)}} \tag{3.16}$$

for $m = m = 2, 3 \ldots\ldots\ldots, M$. By substituting (3.11), (3.15), and (3.16) into (3.13), we finally obtain the following.

For the output units, $m = M - 1$,

$$\delta_j^{(M)} = -e_j \phi_j^{(M)} \left(u_j^{(M)}\right) \tag{3.17}$$

For hidden units, $m = 1, \ldots\ldots\ldots, M-2$,

$$\delta_j^{(M)} = -e_j \phi_j^{(M)} \left(u_j^{(M)}\right) \sum_{\omega=1}^{J_{m+2}} \delta_{\omega}^{(m+2)} \omega_{\omega}^{m+1} \tag{3.18}$$

Equations (3.17) and (3.18) provide a recursive method to solve $\delta_j^{(m+1)}$ for the whole network. Thus, **W** can be adjusted by

$$\frac{\partial E}{\partial \omega_{ij}^{(m)}} = -\delta_j^{(m+1)} o_i^{(m)} \tag{3.19}$$

For the activation functions, we have the following relations

For the logistic function

$$\phi(u) = \beta\phi(u)[1 - \phi(u)] \tag{3.20}$$

For the *tanh* function

$$\phi(u) = \beta\left[1 - \phi^2(u)\right] \tag{3.21}$$

The update for the biases can be in two ways. The biases in the $(m+1)$th layer $\theta^{(m+1)}$ can be expressed as the expansion of the weight $W^{(m)}$, that is, $\theta^{(m+1)} = \left(\omega_{0,1}^{(m)}, \ldots\ldots\ldots\ldots \omega_{0,J_{m+1}}^{(m)}\right)$. Accordingly, the output $\mathbf{o}(m)$ is expanded into $o^{(m)} = \left(1, o_1^{(m)}, \ldots\ldots\ldots, o_{J_m}^{(m)}\right)$. Another way is to use a gradient-descent method with regard to $\boldsymbol{\theta}^{(m)}$, by following the above procedure. Since the biases can be treated as special weights, these are usually omitted in practical applications. The algorithm is convergent in the mean if $0 < \eta < \frac{2}{\lambda_{\max}}$, where $\lambda_{\max}$ is the largest eigenvalue of the autocorrelation of the vector $\mathbf{x}$, denoted as $\mathbf{C}$ [50]. When η is too small, the possibility of getting stuck at a local minimum of the error function is increased. In contrast, the possibility of falling into oscillatory traps is high when η is too large. By statistically preprocessing the input patterns, namely, decorrelating the input patterns, the excessively large eigenvalues of $\mathbf{C}$ can be avoided and thus, increasing η can effectively speed up the convergence. PCA preconditioning speeds up the BP in most cases, except when the pattern set consists of sparse vectors. In practice, η is usually chosen to be $0 < \eta < 1$ so that successive weight changes do not overshoot the minimum of the error surface. The BP algorithm can be improved by adding a momentum term [29]

$$\Delta W(t) = -\eta \frac{\partial E}{\partial W} + \alpha \Delta W(t-1) \qquad (3.22)$$

where α is the momentum factor, usually $0 < \alpha \leq 1$. The typical value for α is 0.9. This method is usually called the *BP with momentum (BPM)* algorithm.

The BP algorithm is a supervised gradient-descent technique, wherein the MSE between the actual output of the network and the desired output is minimized. It is prone to local minima in the cost function. The performance can be improved and the occurrence of local minima reduced by allowing extra hidden units, lowering the gain term, and by training with different initial random weights.

3.3.1 Incremental Learning versus Batch Learning

Incremental learning and batch learning are two methods for the BP learning. For incremental learning, the training patterns are presented to the network sequentially. It is a stochastic optimization method. For each training example, the weights are updated by the gradient-descent method. The learning algorithm has been proved to minimize the global error E when η_{inc} is sufficiently small [29].

In batch learning, the optimization objective is E, and the weight update is performed at the end of an epoch . It is a deterministic optimization method. The weight incremental for each example is accumulated over all the training examples before the weights are actually adapted. For sufficiently small learning rates, incremental learning approaches and batch learning produces the same results [50].

3.3.2 Selecting the Parameters

The performances of the backpropagation algorithm is highly dependent upon a suitable selection for η and α, which, unfortunately, are usually selected by trial-and-error. For different tasks and at different stages of training, some heuristics are needed for optimally adjusting η and α to speed up the convergence of the algorithms. According to Rumelhart [29], the process of starting with a large η and gradually decreasing it is similar to that in simulated annealing [36]. The algorithm escapes from a shallow local minimum in early training and converges into a deeper, possibly global minimum. Learning parameters are typically adapted once for each epoch. All the weights in the network are typically updated using the global learning parameters η and α. The optimal η is the inverse of the largest eigenvalue, *i.e.* $\lambda_{\max}$, of the Hessian matrix **H** of the error function, [37]. An online algorithm for estimating $\lambda_{\max}$ has been proposed by LeCun [51] that does not even require the calculation of the Hessian.

3.3.3 *Backpropagation with Global Descent*

The gradient-descent method is a stochastic dynamical system whose stable points only locally minimize the energy (error) function. The global descent method,[52] which is based on a global optimization technique called terminal repeller unconstrained subenergy tunneling (TRUST) [53, 54], is a deterministic dynamic system consisting of a single vector differential equation. The global descent rule replaces the gradient-descent rule for MLP learning. TRUST was introduced for general optimization problems, and it formulates optimization in terms of the flow of a special deterministic dynamical system.

3.3.4 Robust Backpropagation Algorithms

Since the BP algorithm is a special case of stochastic approximation, the techniques of robust statistics [55] can be applied to the BP. In the presence of outliers-estimator-based robust learning. The rate of convergence is improved since the influence of the outliers is suppressed. Robust BP algorithms using the *M*-estimator-based criterion functions are a typical class of robust algorithms, such as the robust BP using Hampel's tanh estimator with time-varying error cutoff points $\beta 1$ and $\beta 2$, and the annealing robust BP (ARBP) algorithm [56]. The ARBP algorithm adopts the annealing concept into robust learning. A deterministic annealing process is applied to the scale estimator. The algorithm is based on gradient-descent method. As $\beta(t) \rightarrow \infty$, the ARBP becomes a BP algorithm. The basic idea of using an annealing schedule is to use a larger scale estimator in the early training stage and then to use a smaller scale estimator in the later training stage.

3.4 The Fuzzy Logic

The concept of fuzzy sets was first proposed by L.A. Zadeh [57] in the year 1965 as a method for mathematical modeling of the uncertainty in human reasoning. Rather than the binary logic, fuzzy logic uses the notion of membership. Fuzzy logic is most suitable for the representation of hazy data and concepts on an intuitive basis, such as human linguistic description. The conventional or *crisp* set can be treated as a special case of the concept of a fuzzy set. A fuzzy set is uniquely determined by its membership function, and it is also associated with a linguistically meaningful term. Fuzzy logic provides a systematic framework to incorporate human reasoning and experiences. Fuzzy logic is based on three core concepts, namely, fuzzy sets, linguistic variables, and possibility distributions. A fuzzy set is an effective means to represent linguistic variables. A linguistic variable is a variable whose value can be described qualitatively using a linguistic expression and quantitatively using a member function. Linguistic expressions[58] are useful for communicating concepts and knowledge with human beings, whereas membership functions are useful for processing numeric input data. When a fuzzy set is assigned to a linguistic variable, it imposes an elastic constraint, called a *possibility distribution*, on the possible values of the variable. Fuzzy logic is a rigorous mathematical discipline and fuzzy reasoning is a precise formalism for encoding human knowledge in a mathematical framework. In fuzzy control, human knowledge is codified by means of linguistic IF-THEN rules, which build up a fuzzy inference system(FIS).The fuzzy inference systems (FIS) can approximate arbitrarily, any continuous function on a compact domain quite accurately[59]. Applications of fuzzy logic are found in many contexts from medicine to finance, from human factors to consumer products, from vehicle control to computational linguistics etc. Fuzzy logic is widely used in the industrial practice of advanced information technology such as data analysis, regression, signal and image processing. Like the multilayer perceptron(MLP) and the radial basis function network(RBFN), some fuzzy inference systems (FISs) have universal function approximation capability. These systems can be used in many areas where neural networks are applicable.

The most distinguishing property of fuzzy logic is that it deals with fuzzy propositions, that is, propositions which contain fuzzy variables and fuzzy values, for example, "the temperature is high," "the height is short." The truth values for fuzzy propositions are not TRUE/FALSE only, as is the case in propositional Boolean logic, but include all the grayness between two extreme values. Fuzzy rules deal with fuzzy values such as, "high," "cold," "very low," etc. Those fuzzy concepts are usually represented by their membership functions (MF). A membership function shows the extent to which a value from a domain (also called universe) is included in a fuzzy concept. Fuzzy inference methods based on fuzzy logic can be used successfully. Fuzzy inference takes inputs, applies fuzzy rules, and produces outputs. Inputs to a fuzzy system can be either exact, crisp values, or fuzzy values. Output values from a fuzzy system can be fuzzy, for example, a whole membership function for the inferred fuzzy value; or exact (crisp), a single value is produced on the output. The process of transforming an output membership function into a single value is called defuzzification. The secret for the success of fuzzy systems is that they are easy to implement, easy to maintain, easy to understand, robust, and cheap.

A fuzzy system is defined by three main components:

1. Fuzzy input and output variables, defined by their fuzzy values
2. A set of fuzzy rules
3. Fuzzy inference mechanism

3.4.1 Fuzzy Inference Systems and Fuzzy Controllers

Fuzzy logic-based controllers are popular control systems. The general architecture of a fuzzy controller is depicted in Fig. 3.4. Fuzzy controllers are knowledge based, where knowledge is defined by fuzzy IF-THEN rules. The core of a fuzzy controller is an FIS, in which the data flow involves fuzzification, knowledgebase evaluation, and defuzzification. In an FIS, the knowledge base is comprised of the fuzzy rule base and the database. The database contains the linguistic term sets considered in the linguistic rules and the MFs defining the semantics of the linguistic variables, and information about domains. The rule base contains a collection of linguistic rules that are joined by the ALSO operator. An expert provides his knowledge in the form of linguistic rules. The fuzzification process collects the inputs and then converts them into linguistic values or fuzzy sets. The decision logic, called *fuzzy inference engine*, generates output from the input, and finally the defuzzification process produces a crisp output for control action. FISs are universal approximators capable of performing nonlinear mappings between inputs and outputs. The interpretations of a certain rule and the rule base depend on the

FIS model. The Mamdani [60] and the TSK [61] models are two popular FISs. The Mamdani model is a nonadditive fuzzy model that aggregates the output of fuzzy rules using the maximum operator, while the TSK model is an additive fuzzy model that aggregates the output of rules using the addition operator. Kosko's standa additive model (SAM) [62] is another additive fuzzy model. All these models can be derived from fuzzy graph [63], and are universal approximators [59,64,65,66]. Both neural networks and fuzzy logic can be used to approximate an unknown control function. Neural networks achieve a solution using the learning process, while FISs apply a vague interpolation technique. FISs are appropriate for modeling nonlinear systems whose mathematical models are not available. Unlike neural networks and other numerical models, fuzzy models operate at a level of information granules—fuzzy sets.

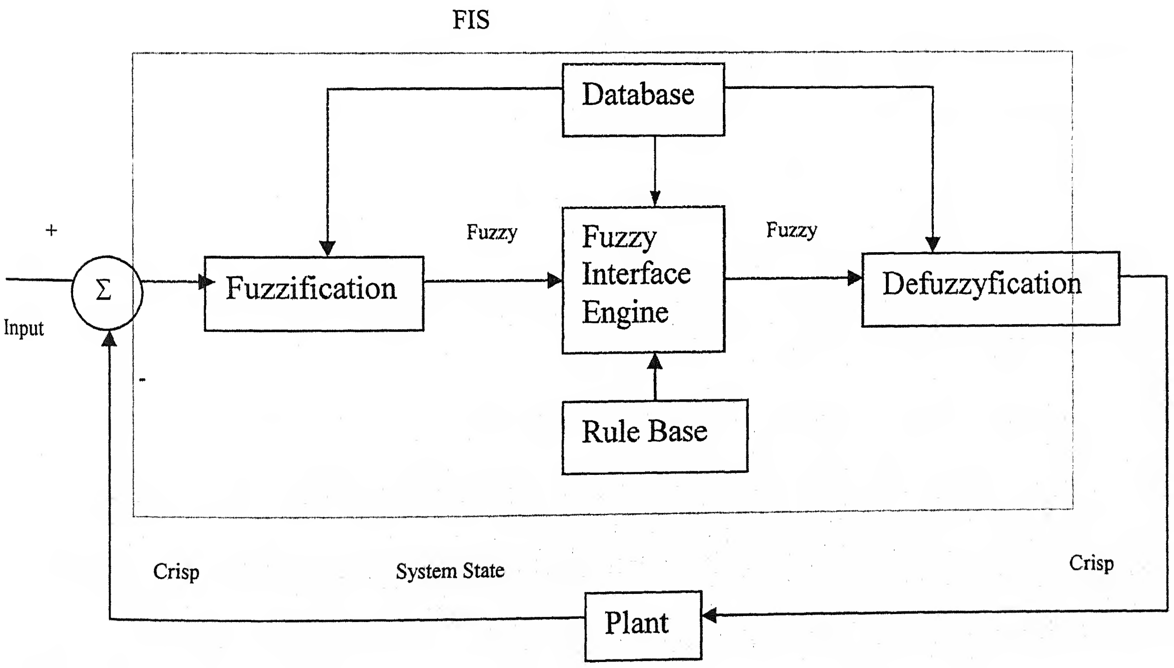


Figure 3.4: ***A model of Fuzzy controller.***

3.4.2 *Fuzzy Rules and Fuzzy Inference*

There are two types of fuzzy rules, namely, *fuzzy mapping rules* and *fuzzy implication rules* [63]. A fuzzy mapping rule describes a functional mapping relationship between inputs and an output using linguistic terms, while a fuzzy implication rule describes a generalized logic implication relationship between two logic formulas involving linguistic variables. Fuzzy implication rules generalize set-to-set implications, whereas fuzzy mapping rules generalize set-to-set associations. The former was motivated to allow intelligent systems to draw plausible conclusions in a way similar to human reasoning, while the latter was motivated to approximate complex relationships such as nonlinear functions in a cost-effective and easily comprehensible way. The foundation of fuzzy mapping rule is fuzzy graph, while the foundation of fuzzy implication rule is a generalization to two-valued logic. A rule base consists of a number of rules in the IF-THEN logic IF *condition*, THEN *action*. The condition, also called *premise*, is made up of a number of *antecedents* that are negated or combined by different operators such as AND or OR computed with t-norms or t-conorms. In a fuzzy-rule system, some variables are linguistic variables and the determination of the MF for each fuzzy subset is critical. MFs can be selected according to human intuition, or by learning from training data. Fuzzy logic can be used as the basis for inference systems. A fuzzy inference is made up of several rules with the same output variables. Given a set of fuzzy rules, the inference result is a combination of the fuzzy values of the conditions and the corresponding actions. For example, we have a set of N_r rules:

$$R_i : IF(condition = C_i) \quad THEN \quad (action = \mathrm{A}_i) \tag{3.23}$$

r $i = 1,2,3,\ldots\ldots\ldots\ldots N_r$ where C_i is a fuzzy set. Assuming that a condition has a nembership degree of μ_i associated with the set C_i. The condition is first converted into fuzzy category using a syntactical representation

$$condition = \frac{C_1}{\mu_1} + \frac{C_2}{\mu_2} + \frac{C_3}{\mu_3} + \ldots\ldots\ldots\ldots \frac{C_{N_r}}{\mu_{N_r}} \tag{3.24}$$

fuzzy inference is the combination of all the possible consequences. The action coming rom a fuzzy inference is also a fuzzy category :

$$action = \frac{\mathrm{A}_1}{\mu_1} + \frac{\mathrm{A}_2}{\mu_2} + \frac{\mathrm{A}_3}{\mu_3} + \ldots\ldots\ldots\ldots \frac{\mathrm{A}_{N_r}}{\mu_{N_r}} \tag{3.25}$$

This is also a syntactical representation. The inference procedure depends on fuzzy easoning. This result can be further processed or transformed into a crisp value.

3.4.3 Fuzzification and Defuzzification

Fuzzification is to transform crisp inputs into fuzzy subsets. Given crisp inputs x_i, where $i = 1,\ldots\ldots\ldots\ldots\ldots n$, fuzzification is to construct the same number of fuzzy sets A^i

$$\mathrm{A}^i = fuzz(x_i), \tag{3.26}$$

where fuzz( ) is a fuzzification operator. Fuzzification is determined according to the defined MFs.

Defuzzification is to map fuzzy subsets of real numbers into real numbers. In an FIS, defuzzification is applied after aggregation. Defuzzification is necessary in fuzzy ontrollers, since the machines cannot understand control signals in the form of a omplete fuzzy set. Popular defuzzification methods include the centroid defuzzifier [54], nd the mean-of-maxima defuzzifier [60]. The centroid defuzzifier is the bestknown nethod, which is to find the centroid of the area surrounded by the MF and the horizontal xis. A discrete centroid defuzzifier is given by [67]

$$defuzz(B) = \frac{\sum_{i=1}^{K} \mu B(Y_i) Y_i}{\sum_{i=1}^{K} \mu B(Y_i)} \tag{3.26}$$

vhere K is the number of quantization steps by which the universe of discourse Y of the MF $\mu B(y)$ is discretized. Aggregation and defuzzification can be combined into a single phase, such as the weighted-mean method [68]

$$defuzz(B) = \frac{\sum_{i=1}^{N_r} \mu_i b_i}{\sum_{i=1}^{N_r} \mu_i} \tag{3.27}$$

where Nr is the number of rules, μ_i is the degree of activation of the i^{th} rule, and b_i is a numerical value associated with the consequent of the i^{th} rule, B_i.The parameter b_i can be selected as the mean value of the α-level set when α is equal to μ_i [68].

3.4.4 Complex Fuzzy Logic

Complex fuzzy sets and logic are mathematical extensions of fuzzy sets and logic from the real domain to the complex domain [69,70]. A complex fuzzy set S is characterized by a complex-valued MF, and the membership degree of any element x in S is given by a complex value of the form

$$\mu_S(x) = r_S(x)e^{i\varphi_S(x)} \tag{3.28}$$

where the amplitude $r_S(x) \in [0,1]$, and φ_S is the phase, that is, $\mu_S(x)$ is within a unit circle in the complex plane. As in the development of the fuzzy logic theory, the design of the settheoretic operations is of vital importance in the complex fuzzy logic. The basic set operators for fuzzy logic have been extended for the complex fuzzy logic, and some additional operators such as the vector aggregation, set rotation and set reflection, are also defined [69,70]. The operations of intersection, union and complement for complex fuzzy sets are defined on the modulus of the complex membership degree without a consideration of its phase information. The complex fuzzy logic is extended to logic of vectors in the plane, rather than scalar quantities. A complex fuzzy set is defined as an MF mapping the complex plane into [0, 1] × [0, 1]. Complex fuzzy sets are superior to the Cartesian products of two fuzzy sets. Complex fuzzy logic maintains both the advantages of the fuzzy logic and the properties of complex fuzzy sets. In complex fuzzy logic, rules constructed are strongly related and a relation manifested in the phase term is associated with complex fuzzy implications. In a complex FIS, the output of each rule is a complex fuzzy set, and phase terms are necessary when combining multiple rules so as to generate the final output. Complex FISs are useful for solving problems in which rules are related to one another with the nature of the relation varying as a function of the input to the system [68]. These problems may be very difficult or impossible to solve using traditional fuzzy methods. The fuzzy complex number [69] is a different concept from the complex fuzzy set [70]. The fuzzy complex number was introduced by incorporating the complex number into the support of the fuzzy set. A fuzzy complex number is a fuzzy set of complex numbers, which have real-valued membership degree in the range [0, 1]. An α-cut of a fuzzy complex number is based on the modulus of the complex numbers in the fuzzy set. The operations of addition, subtraction, multiplication and division for fuzzy complex numbers are derived using the extension principle, and closure of the set of fuzzy complex numbers is proved under each of these operators. In a nutshell, a fuzzy complex number is a fuzzy set in one dimension, while a complex fuzzy set or number is a fuzzy set in two dimensions.

3.5 The Evolutionary Algorithm

Evolutionary algorithms are a class of general-purpose stochastic optimization algorithms under the universally accepted neo-Darwinian paradigm. The neo- Darwinian paradigm is a combination of the classical Darwinian evolutionary theory, the selectionism of Weismann, and the genetics of Mendel [71]. EAs are currently a major approach to adaptation and optimization. Evolutionary algorithms are also known as the optimum search methods that take their inspiration from natural selection and survival of the fittest in the biological world. EAs differ from more traditional optimization techniques in that they involve a search from a "population" of solutions, not from a single point. Each iteration of an EA involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen. An extended discussion of issues involved with the implementation and use of evolutionary algorithms is included here. Several different types of evolutionary algorithms were developed independently. These include genetic programming (GP), which evolve programs, evolutionary programming (EP), which focuses on optimizing continuous functions without recombination, evolutionary strategies (ES), which focuses on optimizing continuous functions with recombination, and genetic algorithms (GAs) [72], which focuses on optimizing general combinatorial problems.

For an optimization problem in a domain, if the calculus is difficult to implement or is inapplicable, search methods such as EAs can be used. EAs are a class of stochastic search and optimization techniques guided obtained by natural selection and genetics. They are population-based algorithms by simulating the natural evolution of biological systems. Individuals in a population compete and exchange information with one another. There are three basic genetic operations, namely, *crossover*, *mutation*, and *selection*. EAs are stochastic processes performing searches over a complex and multimode space. They have the several advantages such as:

- The EA approach is a general-purpose one that can be directly interfaced to existing simulations and models.
- They are suitable for evaluation functions that are large, complex, non-continuous, non-differentiable, and multimodal.
- They are extendable and easy to hybridize so that they can reach the near optimum or the global maximum.

- Evolutionary algorithms possess inherent parallelism by evaluating multipoint simultaneously also employ a structured, yet randomized, parallel multipoint search strategy that is biased toward reinforcing search points of high fitness.

3.5.1 The Terminologies Used

To define the evolutionary algorithm, the following terminologies analogous to biological counterparts are used.

Population

A set of individuals in a generation is called a *population*, $P(t)=\{x_1,x_2,.....x_{N_P}\}$, where x_i is the i^{th} individual, and N_P is the size of the population. The initial populations are usually generated randomly, while the population of other generations is generated from some selection and/or reproduction procedure.

Chromosome

Each individual x_i in a population is a single *chromosome*. A chromosome, sometimes called a *genome*, is a set of parameters that define a solution to the problem under consideration. The chromosome is often represented as a string in EAs. Biologically, a chromosome is a long, continuous piece of DNA that contains many genes, regulatory elements and other intervening nucleotide sequences.

Gene

In evolutionary algorithms, each chromosome X comprises of a string of elements x_i, called *genes*, $i.e.\ X=[x_1,x_2,..........x_n]$, where n is the number of genes in the chromosome. Each gene encodes a parameter of the problem into the chromosome. A gene is usually encoded as a binary string or a real number. In biology, genes are entities that parents pass to offspring during reproduction.

Allele

The biological definition for an *allele* is any one of a number of alternative forms of the same gene occupying a given position called a *locus* on a chromosome. In the EA terminology, the value of a gene is indicated as an *allele*.

Genotype

A *genotype* is biologically referred to the underlying genetic coding of a living organism, usually in the form of DNA. The genotype of each organism corresponds to an observable, known as a *phenotype*. In evolutionary algorithms, a genotype represents a coded solution, that is, an individual's chromosome.

Phenotype

Biologically, the *phenotype* of an organism is either its total physical appearance and constitution or a specific manifestation of a trait. A phenotype is determined by genotype or multiple genes and influenced by environmental factors. The concept of phenotypic plasticity describes the degree to which an organism's phenotype is determined by its genotype. A high level of plasticity means that environmental factors have a strong influence on the particular phenotype that develops. The ability to learn is the most obvious example of phenotypic plasticity. In EAs, a phenotype represents a decoded solution.

Fitness

Fitness in biology refers to the ability of an individual of certain genotype to reproduce. The set of all possible genotypes and their respective fitness values is called a *fitness landscape*. Fitness function is a particular type of objective function that quantifies the optimality of a solution, *i.e.* a chromosome, in an EA. It is used to map an individual's chromosome into a positive number. Fitness is the value of the objective function for a chromosome x_i, namely $f(x_i)$. After the genotype is decoded, the fitness function is used to convert the phenotype's parameter values into the fitness. Fitness is used to rate the solutions.

Natural Selection

Natural selection is believed to be the most important mechanism in the evolution of biological species. It alters biological populations over time by propagating heritable traits affecting individual organisms to survive and reproduce. It is concerned with those traits that help individuals to survive the environment and to reproduce. Natural selection is different from artificial selection. Genetic drift and gene flow are two other mechanisms in biological evolution. Genetic flow, also known as *genetic migration*, is the migration of genes from one population to another.

Genetic Drift

Genetic drift is a contributing mechanism in biological evolution. As opposed to natural selection, genetic drift is a stochastic process that arises from random sampling in the reproduction. It changes allele frequencies (gene variations) in a population over many generations and affects traits that are more neutral. The genes of a new generation are a sampling from the genes of the successful individuals of the previous one, but with some statistical error. Drift is the cumulative effect over time of this sampling error on the allele frequencies in the population, and traits that do not affect reproductive fitness change in a population over time. Like selection, genetic drift acts on populations, altering allele frequencies and the predominance of traits. It occurs most rapidly in small populations and can lead some alleles to become extinct or become the only alleles in the population, thus reducing the genetic diversity in the population.

Termination Criterion

The search process of an EA will terminate when a certain termination criterion is met. Otherwise a new generation will be produced and the search process continues. The termination criterion can be selected as a maximum number of generations, or the convergence of the genotypes of the individuals. Convergence of the genotypes occurs when all the bits or values in the same positions of all the strings are identical, and crossover has no effect for further processes. Phenotypic convergence without genotypic convergence is also possible. For a given system, the objective values are required to be mapped into fitness values so that the domain of the fitness function is always greater than zero.

3.5.2 The Genetic Algorithm

Genetic Algorithms are unorthodox search or optimization algorithms, which were first suggested by John Holland in 1975. The GA [73] is the most popular form of EAs. Concisely stated, a genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a *fitness function* that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random. The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem. These promising candidates are kept and allowed to reproduce. Multiple populations are made of them, but the populations are not perfect; random changes are introduced during the process of population generation. These digital offspring then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. Those candidate solutions which were worsened, or made no better, by the changes to their code are again deleted; but again, purely by chance, the random variations introduced into the population may have improved some individuals, making them into better, more complete or more efficient solutions to the problem at hand. Again these winning individuals are selected and copied over into the next generation with random changes, and the process repeats. The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered.

As astonishing and counterintuitive as it may seem to some, genetic algorithms have proven to be an enormously powerful and successful problem-solving strategy, dramatically demonstrating the power of evolutionary principles. Genetic algorithms have been used in a wide variety of fields to evolve solutions to problems as difficult as or more difficult than those faced by human designers. Moreover, the solutions they come up with are often more efficient, more elegant, or more complex than anything comparable a human engineer would produce. In some cases, genetic algorithms have come up with solutions that baffle the programmers who wrote the algorithms in the first place.

3.5.2.1 Representation or Encoding/Decoding

The GA uses binary coding. A chromosome **x** is a potential solution, denoted by a concatenation of the parameters $X = [x_1, x_2, ...x_n]$ where each x_i is a gene, and the value of x_i is an allele *X is encoded* as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space"

A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution. One example of this technique is Hiroaki Kitano's "grammatical encoding" approach, where a GA was put to the task of evolving a simple set of rules called a context-free grammar that was in turn used to generate neural networks for a variety of problems [74].

Another strategy, developed principally by John Koza [75]of Stanford University and called *genetic programming*, represents programs as branching data structures called trees. In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one sub- tree with another.

3.5.2.2 Selection or Reproduction

Selection embodies the principle of *survival of the fittest*, which provides a driving force in the GA. Selection is based on the fitness of the individuals. There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation, but listed below are some of the most common methods. Some of these methods are mutually exclusive, but others can be and often are used in combination.

Elitist selection: The fit members of each generation are guaranteed to be selected. It is the most commonly used technique, is also known as elitism. The elitism strategy for selecting the individual with best fitness can improve the convergence of the GA [74]. The elitism strategy always copies the best individual of a generation to the next generation. Although elitism may increase the possibility of premature convergence, it improves the performance of the GA in most cases and thus, is integrated in most GA implementations [76].

Fitness-proportionate selection: More fit individuals are more likely, but not certain, to be selected.

Roulette-wheel selection: A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness.

Scaling selection: As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating.

Tournament selection: Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

Rank selection: Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute differences in fitness.
Generational selection: The offspring of the individuals selected from each generation become the entire next generation. No individuals are retained between generations.

Steady-state selection: The offspring of the individuals selected from each generation go back into the pre-existing gene pool, replacing some of the less fit members of the previous generation. Some individuals are retained between generations.

Hierarchical selection: Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously.

3.5.2.3 Replacement or Change

During the selection procedure, we need to decide as to how many individuals in one popu2lation will be replaced by the newly generated individuals so as to produce the population for the new generation. Thus, the selection mechanism is split into two phases, namely, parental selection and replacement strategy. There are many replacement strategies such as the complete generational replacement [77], replace-random, replace-worst, replaceoldest, and deletion by kill tournament [78]. In the crowding strategy , an offspring replaces one of the parents whom it most resembles using the similarity measure of the Hamming distance. These replacement strategies may result in a situation where the best individuals in a generation may fail to reproduce. In this problem is solved by introducing into the system a new variable that stores the best individuals obtained so far. Elitism strategy cures the same problem without changing the system state . There are two basic strategies to accomplish this. The first and simplest is called *mutation and the other one is crossover*.

Both crossover and mutation are considered the driving forces of evolution. Crossover occurs when two parent chromosomes, normally two homologous instances of the same chromosome, break and then reconnect but to the different end pieces. Mutations can be caused by copying errors in the genetic material during cell division and by external environment factors. Although the overwhelming majority of mutations have no real effect, some can cause disease in organisms due to partially or fully nonfunctional proteins arising from the errors in the protein sequence.

Crossover is the primary exploration operator in the GA, which searches the range of possible solutions based on existing solutions. Crossover, as a binary operator, is to exchange information between two selected parent chromosomes at randomly selected positions and to produce two new offspring (individuals). Both the children will be different from either of their parents, yet retain some features of both. The method of crossover is highly dependent on the method of the genetic coding. Some of the commonly used crossover [79] techniques are the one-point crossover, the two-point crossover, the multipoint crossover, and the uniform crossover. The crossover points are typically at the same, random positions for both parent chromosomes. These crossover operators are illustrated in Figure 3.5.

Parents	A	B	C	D	E	F	G	H	I	J
	a	b	c	d	e	f	g	h	i	j

Children	A	b	C	D	e	F	g	h	I	j
	a	B	c	d	E	f	G	H	i	J

Figure 3.5: Generalized crossover.

The one-point crossover requires one crossover point on the parent chromosomes, and all the data beyond that point are swapped between the two parent chromosomes. The one-point crossover is easy to model analytically, and it generates bias toward bits at the ends of the strings, everything between the two points is swapped. ***The two-point crossover*** causes a smaller schema disruption than the one-point crossover. The two-point crossover eliminates this disadvantage of the one-point crossover, but generates bias at a different level. *Multipoint crossover* treats each string as a ring of bits divided by *m* crossover points into *m* segments, and each segment is exchanged at a fixed probability. *Uniform Crossover* exchanges bits of a string rather than segments. Individual bits in the parent chromosomes are compared, and each of the non-matching bits is probabilistically swapped with a fixed probability, typically 0.5. The uniform crossover is unbiased with respect to defining length. In the half-uniform crossover, exactly half of the nonmatching bits are swapped.

Mutation

Mutation is a unary operator that requires only one parent to generate an offspring. A mutation operator typically selects a random position of a random chromosome and replaces the corresponding gene or bit by other information. Mutation helps to regain the lost alleles into the population. Mutations can be classified into point mutations and large-scale mutations. Point mutations are changes to a single position, which can be substitutions, deletions, or insertions of a gene or a bit. Large-scale mutations can be similar to the point mutations, but operate at multiple positions simultaneously, or at one point with multiple genes or bits, or even on the chromosome scale. Functionally, mutations introduce the necessary amount of noise to do hill climbing. Two additional large-scale mutation operators are the inversion and rearrangement operators. The inversion operator [73] picks up a portion between two randomly selected positions within a chromosome and then reverses it. Inversion reshuffles the order of the genes in order to achieve a better evolutionary potential. However, its computational advantage over other conventional genetic operators is not clear. The swap operator is the most primitive reordering operator; based on which many new unary operators including inversion can be derived. The rearrangement operator reshuffles a portion of a chromosome such that the juxtaposition of the genes or bits is changed. Some mutation operations are illustrated in Figure 3.6.

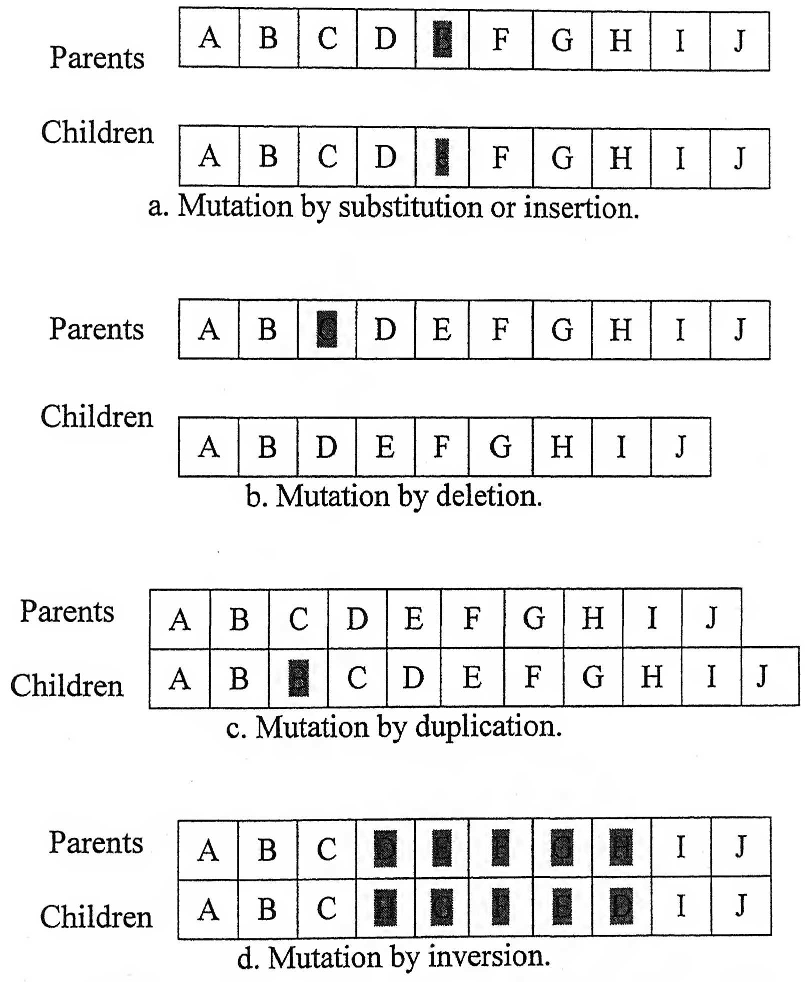


Figure 3.6: ***Different types of mutation. The alphabets show the bits or gene.***

A high mutation rate can lead genetic search to random search. A high mutation rate may change the value of an important bit, and thus slow down the fast convergence of a good solution or slow down the process of convergence of the final stage of the iterations. Thus, mutation is made occasionally in the GA. In the simple GA, the mutation is typically selected as a substitution operation that changes one random bit in the chromosome at a time. An empirically derived formula that can be used as the probability of mutation P_m at a starting point is given by:

$$P_m = \frac{1}{T\sqrt{l}} \tag{3.29}$$

where T is the total number of generations and l is the string length. The random nature of mutation and its low probability of occurrence make the convergence of the GA slow. The search process can be expedited by using the directed mutation technique [80] that deterministically introduces new points into the population by using gradient or extrapolation of the information acquired so far. In passing, it may be mentioned that the relative importance of crossover and mutation has been discussed in the GA community and no compelling conclusion has been drawn. It is commonly agreed that crossover plays a more important role if the population size is large, and mutation is more important if the population size is small.

3.6 Immune Algorithms

In 1974, Jerne [81] proposed a network theory for the immune system based on the clonal selection theory [82]. The biological immune system has the features of immunological memory and immunological tolerance. It has self-protection and self-regulation mechanisms. The basic components of the immune system are two types of lymphocytes, namely B-lymphocytes and Tlymphocytes, which are cells produced by bone marrow and by the thymus, respectively. B-lymphocytes generate antibodies on their surfaces to resist their specific antigens, while T-lymphocytes regulate the production of antibodies from B-lymphocytes. Roughly 107 distinct types of B-lymphocytes exist in a human body. An antibody recognizes and eliminates a specific type of antigen. The clonal selection theory describes the basic features of an immune response to an antigenic stimulus. The clonal operation is an antibody artificial immune networks [83] employ two types of dynamics. The short-term dynamics govern the increase or decrease of the concentration of a fixed set of lymphocyte clones and the corresponding immunologist. The metadynamics govern the recruitment of new species from an enormous pool of lymphocytes freshly produced by the bone marrow. The short-term dynamics correspond to a set of cooperating or competing agents, while the metadynamics refine the results of the short-term dynamics. As a result, the short-term dynamics are closely related to neural networks and the metadynamics are similar to the GA. The immune algorithm, also called the *clonal selection algorithm*, introduces suppress cells to change search scope and memory cells to keep the candidate solutions. It is an EA inspired by the immune system, and is very similar to the GA. In the immune algorithm, *antigen* is defined as the problem to be optimized, and *antibody* is the solution to the objective function. Only those lymphocytes that recognize the antigens are selected to proliferate. The selected lymphocytes are subject to an affinity maturation process, which improves their affinity to the selective antigens. Learning in the immune system involves raising the relative population size and affinity of those lymphocytes. The immune algorithm first recognizes the antigen, and produces antibodies from memory cells. Then it calculates the affinity between antibodies, which can be treated as fitness. Antibodies are dispersed to the memory cell, and the concentration of antibodies is controlled by stimulating or suppressing antibodies. A diversity of antibodies for capturing unknown antigen is generated using genetic reproduction operators. The immune mechanism can also be defined as a genetic operator and integrated into the GA [84]. The immune operator overcomes the blindness in action of the crossover and mutation and to make the fitness of population increase steadily. The immune operator is composed of two operations, namely a vaccination and an immune selection, and it utilizes reasonably selected vaccines to intervene in the variation of genes in an individual chromosome.

3.7 Ant-colony Optimization

The ACO [85-88] is a metaheuristic approach for solving discrete or continuous optimization problems such as COPs. The ACO heuristic was inspired by the foraging behavior of ants. Ants are capable of finding the shortest path between the food and the colony (nest) due to a simple pheromone-laying mechanism. The optimization is the result of the collective work of all the ants in the colony. Ants use their pheromone trails as a medium for communicating information. All the ants contribute to the pheromone reinforcement. Old trails will vanish due to evaporation. Different ACO algorithms arise from different pheromone value update rules. The ant system [85] is an evolutionary approach, where several generations of artificial ants search for good solutions. Every ant of a generation builds up a complete solution, step by step, going through several decisions by choosing the nodes on a graph according to a probabilistic *state transition* *rule*, called the *random-proportional rule*. The probability for ant k at node I moving to node j at generation t is defined by

$$P_{i,j}^{k} = \frac{\tau_{i,j}(t) d_{i,j}^{-\beta}}{\sum_{u \in J_i^k} \tau_{i,u} d_{i,u}^{-\beta}} \tag{3.30}$$

for $j \in J_i^k$,where $\tau_{i,j}$ is the intensity of the pheromone on edge $i \rightarrow j$, $d_{i,j}$ is the distance between nodes i and j, J_i^k is the set of nodes that remain to be visited by ant k positioned at node i to make the solution feasible, and $\beta > 0$ is a parameter that determines the relative importance of pheromone vs. distance. A tabu list is used to save the nodes already visited during each generation. When a tour is completed, the tabu list is used to compute the ant's current solution. Once all the ants have built their tours, the pheromone is updated on all edges $i \rightarrow j$ according to a *global-pheromone updating rule*

$$\tau_{i,j}(t+1) = (1|-\alpha)\tau_{i,j}(t) + \sum_{k=1}^{N_P} \tau_{i,j}^{k}(t) \tag{3.31}$$

where $\tau_{i,j}^k$ is the intensity of the pheromone on edge $i \rightarrow j$ contributed by ant k, taking $\frac{1}{L_k}$ if $i \rightarrow j$ is an edge used by ant k, and 0 otherwise, $\alpha \in (0,1)$ is a pheromone decay parameter, L_k is the length of the tour performed by ant k, and N_P is the number of ants. Thus, a shorter tour gets a higher reinforcement. Each edge has an LTM to store the pheromone.The ant-colony system (ACS) [85] improves the ant system by applying a *local pheromone updating rule* during the construction of a solution. The global updating rule is applied only to edges that belong to the best ant tour. Some important subsets of

ACO algorithms[86], such as the most successful ACS and min-max ant system (MMAS) algorithms, have been proved to converge to the global optimum [87].

3.8 Hybrid Approaches

As far as the computation speed is concerned, it is hard to say whether EAs can compete with the gradient-descent method or not. There is no clear winner in terms of the best training algorithm, since the best method is always problem dependent. For large networks, EAs may be inefficient. When gradient information is readily available, it can be used to speed up the evolutionary search. The process of learning facilitates the process of evolution. The hybrid of evolution and gradient search is an effective alternative to the gradient-descent method in learning tasks, when the global optimum is at a premium. The hybrid method is more efficient than either an EA or the gradient-descent method used alone. As is well known, EAs are inefficient in fine tuning local search although they are good at global search. This is especially true for the GA. By incorporating a local-search procedure such as the gradient descent into the evolution, the efficiency of evolutionary training can be improved significantly. Neural networks can be trained by alternating two steps, where an EA step is first used to locate a near-optimal region in the weight space and a local-search solution in that region [89]. Hybridization of EAs and local search can be based either on the Lamarckian strategy or on the Baldwin effect. Local search corresponds to the phenotypic plasticity in biological evolution. Since Hinton and Nowlan constructed the first computational model of the Baldwin effect in 1987 [90]. The hybrid methods based on the Lamarckian strategy and the Baldwin effect are very successful with numerous implementations. Although the Lamarckian evolution is biologically implausible, it has proved effective within computer applications. Nevertheless, the Lamarckian strategy has been pointed out to distort the population so that the schema theorem no longer applies [91]. The Baldwin effect only alters the fitness landscape and the basic evolutionary mechanism remains purely Darwinian. Thus, the schema theorem still applies to the Baldwin effect [91].

3.8.1 Fuzzy Systems with Evolutionary Algorithms

FISs are highly nonlinear systems with many input and output variables. A crucial issue in FISs is the generation of fuzzy rules. EAs can be employed with Evolutionary Algorithms and Evolving Neural Networks for generating fuzzy rules and adjusting MFs of fuzzy sets. Sufficient system information must be encoded and the representation must be easy for evaluation and reproduction. A fuzzy rule base can be evolved by encoding the number of rules and the MFs comprising those rules into one chromosome. All the input and output variables and their corresponding MFs in the fuzzy rules are encoded. The genetic coding of each rule is the concatenation of the shape and location parameters of the MFs of all the variables. If some rules in two individuals are ordered in different manners, the rules should be aligned before reproduction. This leads to much less time for evolution [92]. In addition to existing genetic operators, specific genetic operators such as rule insertion or rule deletion, where a whole rule is added or deleted at a specified point of the string, are also applied [92]. In automatic optimal design of fuzzy systems is conducted by using the GESA [93]. The ES approach is more suitable for the design of fuzzy systems due to their direct coding for real parameter optimization. Consequently, the length of the objective vector increases linearly with the number of variables. Conversely, when the GA is employed, all the parameters need to be converted into fixed-length binary strings.

3.8.2 Neural Networks with Evolutionary Algorithms

Neural-network learning is a search process for the minimization of a criterion or error function. In order to make use of existing learning algorithms, one needs to select a lot of parameters, such as the number of hidden layers, the number of units in each hidden layer, the type of learning rule, the transfer function, as well as learning parameters. In general, one usually selects an effective architecture by hand, and thus the procedure is time consuming. Moreover, gradient-based algorithms usually run multiple times to avoid local minima and also gradient information must be available. There are also adaptive methods for automatically constructing neural networks such as the upstart algorithm [94] and the growing neural tree method [95]. Evolution can be introduced into neural networks at many different levels. The evolution of connection weights provides a global approach to connection weight training. When EAs are used to construct neural networks, a drastic reduction in development time and simpler designs can be achieved. The optimization capability of EAs can lead to a minimal configuration that reduces the total training time as well as the performing time for new patterns. In most cases, an individual in an EA is selected as a whole network. Competition occurs among those individual networks, based on the performance of each network. Within this framework, an EA can be used to evolve the structure, the parameters, and/or the nonlinear activation function of a neural network. A recent survey on evolving neural networks is given in [96]. There are also occasions when EAs are used to evolve one hidden unit of the network at a time. The competing units are individual units. During each successive run, candidate hidden units compete so that the optimal single unit is added in that run. The search space of EAs at each run is much smaller. However, the entire set of hidden units may not be the global optimal placement.

Hybrid Training

As far as the computation speed is concerned, it is hard to say whether EAs can compete with the gradient-descent method or not. There is no clear winner in terms of the best training algorithm, since the best method is always problem dependent. For large networks, EAs may be inefficient. When gradient information is readily available, it can be used to speed up the evolutionary search. The process of learning facilitates the process of evolution. The hybrid of evolution and gradient search is an effective alternative to the gradient-descent method in learning tasks, when the global optimum is at a premium. The hybrid method is more efficient than either an EA or the gradient-descent method used alone. As is well known, EAs are inefficient in fine tuning local search although they are good at global search. This is especially true for the GA. By incorporating a local-search procedure such as the gradient descent into the evolution, the efficiency of evolutionary training can be improved significantly. Neural networks can be trained by alternating two steps, where an EA step is first used to locate a near-optimal region in the weight space and a local-search step such as the gradient-descent step is then used to find a local-optimal solution in that region [88,97]. Hybridization of EAs and local search can be based either on the Lamarckian strategy or on the Baldwin effect. The hybrid methods based on the Lamarckian strategy and the Baldwin effect are very successful with numerous implementations. The Baldwin effect only alters the fitness landscape and the basic evolutionary mechanism remains purely Darwinian. Thus, the schema theorem still applies to the Baldwin effect [91].

Evolving Network Parameters

EAs are robust search and optimization techniques, and can locate the near global optimum in a multimodal landscape. They can be used to optimize neural-network structure and parameters, or to optimize specific network performance and algorithmic parameters. EAs are suitable for learning networks with non differentiable activation function. Considerable research has been conducted on the evolution of connection weights, and the references therein. EAs evolve network parameters such as weights based on a fitness measure for the whole network. The fitness function can usually be defined as $\frac{1}{1+E}$ where E is the error or criterion function for network training. The complete set of network parameters is coded as a chromosome s with a fitness function $f(s)=\frac{1}{1+E(D(s))}$, where D(s) is a decoding transformation.

Coding of network parameters is most important from the point of view of the convergence speed of search. When using the binary GA, the fixed-point coding is shown to be usually superior to the floating-point coding of the parameters [97]. For crossover, it is usually better to only exchange the parameters between two chromosomes, but not to change the bits of each parameter [97]. The modifications of network parameters can be conducted by mutation. Due to the limitation of the binary coding, real numbers are usually used to represent network parameters directly [98]. Each individual is a real vector, and crossover and mutation are specially defined for real-coded EAs.

The architecture of a neural network is referred to as its topological structure, *i.e. connectivity*. The network architecture is usually predefined and fixed. Design of the optimal architecture can be treated as a search problem in the architecture space, where each point represents architecture. Given certain performance criteria, such as minimal training error and lowest network complexity, the performance levels of all architectures form a discrete surface in the space. The performance surface is no differentiable due to a discrete number of nodes, and multimodal since different architectures have similar performance. Direct and indirect encodings are two methods for encoding architecture. For the direct encoding, every connection of the architecture is encoded into the chromosome. For the indirect encoding, only the most important parameters of an architecture, such as the number of hidden layers and the number of hidden units in each hidden layer, are encoded. Only the architecture of a network is evolved, whereas other parameters of the architecture such as the connection weights have to be learned after a near-optimal architecture is found.

One major problem with the evolution of architectures without connection weights is noisy fitness evaluation [96]. The noise is dependent on the random initialization of the weights and the training algorithm used. The noise identified is caused by the one-to-many mapping from genotypes to phenotypes. Thus, the evolution of architectures without any weight information is inaccurate for evaluating fitness, and the evolution would be very inefficient. In [99], an improved GA is used for training a three-layer FNN with switches at its links. Both the nonlinear mapping and the network architecture can be learned. The weights of the links govern the input-output mapping, while the switches of the links govern the network architecture. In the genetic backpropagation (G-Prop) method [100], the GA selects the initial weights and changes the number of neurons in the hidden layer through the application of five specific genetic operators, namely, mutation, multipoint crossover, addition, elimination and substitution of hidden units. The BP, on the other hand, is used to train from these weights. This makes a clean division between global and local search. This strategy attempts to avoid Lamarckism. Behaviors between parents and their offspring are linked by various mutations, such as partial training and node splitting. The evolved neural network is parsimonious by preferring the node/connection deletion operations to the node/connection addition operations. The hybrid training operator that consists of a modified BP with adaptive learning rates and the SA is used for modifying the connection weights. After the evolution, the best evolved neural network is further trained using the modified BP on the combined training and validation set.

There are also studies on evolving node activation functions or learning rules, since different activation functions have different properties and different learning rules have different performance [96]. For example, the learning rate and the momentum factor of the BP algorithm can be evolved [101], and learning rules evolved to generate new learning rules [102]. EAs are also used to select proper input variables for neural networks from a raw data space of a large dimension, that is, to evolve input features.

The activation functions can be evolved by selecting among some popular nonlinear functions such as the Heaviside, sigmoidal, and Gaussian functions. A neural network with evolutionary neurons has been proposed in [103].

3.9 Conclusion

In this chapter we have explored the different aspects of soft computing methods. The potential & applicability of the soft computing techniques were discussed in detail. The discussion has started from the artificial neural network architecture and their various learning algorithms in order to train the network for specified task. The backpropagation algorithm has been discussed with its capabilities and limitations. The different modifications of backpropagation algorithm proposed by the researchers were also discussed in detail.

In the continuation the fuzzy logic, fuzzy rules, fuzzy inference system has been discussed for accomplish the task of pattern recognition with its ability to deal with impreciseness, incompleteness and uncertainty in the input stimuli. The advance fuzzy systems are also explored like complex fuzzy system. The discussion proceeds next with the evolutionary algorithm. The immune algorithm & ant colony optimization has also been discussed as the new trends of evolutionary learning to determine the global optimal solution. In the last the hybrid evolutionary architecture and algorithm has explained. The various methods which incorporate the genetic algorithm with neural networks have discussed with their advantages & efficiency in order to find the global optimum solution.

References

[1] Antognetti, P. and Milutinovic, V., "Neural Networks: Concepts, Applications, and Implementations (Eds.)", vol. I-IV, Prentice Hall, Englewood Cliffs, NJ., (1991).

[2] Anderson, J.A. and Rosenfeld, E., (eds.), "Neurocomputing: Foundations of Research." MIT Press, Boston, MA, (1988).

[3] Robert,H.N., "Neurocomputing", Addison – Wesley, ISBN 0-201-09255-3, (1990).

[4] Stanely and Jeannette, "Introduction to Neural Networks", Californian Scientific Software, (1988).

[5] Goldberg, D., "Genetic Algorithms in Search, Optimization, and Machine Learning Reading", MA: Addison-Wesley Publishing Company, Inc., (1989).

[6] Hoffman, "Arguments on Evolution: A Paleontologist's Perspective", New York: Oxford University Press, (1988).

[7] Malthus, T.R., "An Essay on the Principle of Population, as it Affects the Future Improvement of Society", 6th ed., London: Murray, (1826).

[8] Mayr, E., "The Growth of Biological Thought: Diversity, Evolution and Inheritance", Cambridge, MA: Belknap Press, (1988).

[9] Wright, S., "Evolution in Mendelian Populations," Genetics, vol. 16, pp. 97-159, (1931).

[10] Wright, S., "The evolution of life: Panel discussion in Evolution After Darwin: Issues in Evolution.", vol. III, S. Tax and C. Callender, Eds. Chicago: Univ. of Chicago Press, (1960).

[11] Bremermann, H.J., "The Evolution of Intelligence. The Nervous System as A Model of Its Environment." Technical Report No. 1, Contract No. 477(17). Dept. of Mathematics, Univ. of Washington, Seattle, (1958).

[12] Bremermann, H. J., "Optimization through Evolution and Recombination," in Self Organizing Systems. M. C. Yovits, G. T. Jacobi, and G. D. Goldstine, Eds. Washington, DC: Spartan Books, pp. 93-106, (1962).

[13] Bremermann, H. J. and Rogson, M., "An Evolution-Type Search Method for Convex Sets," ONR Technical Report, Contracts 222(85) and 3656(58), UC Berkeley, (1964).

[14] Bremermann H. J., Rogson,M. and Salaff,S.,"Search by Evolution," in Biophysics and Cybernetic Systems. M. Maxfield, A. Callahan, and L. J. Fogel, Eds. Washington, DC: Spartan Books, pp. 1.57-167, (1965).

[15] Hornik,K., Stinchcombe,M. and White,H. ,"Approximation capabilities of multiplayer feedforward networks", Neural Networks, Vol. 4, pp. 251-257, (1989).

[16] Atlan,H. and Cohen,I.R. "Theories of immune networks. Spriner-Verlag, Berlin,(1989),

[17] Reed,J., Toombs,R. and Banicelli, N. A., "Simulation of Biological Evolution and Machine Learning," Journal of Theoretical Biology, vol. 17, pp. 319-342, (1967).

[18] Ho,Y.C. and Kashyap, R.L., "An algorithm for linear inequalities and its applications", IEEE Transactions on Electronic Computers, vol. 14, pp. 683-688.

[19] Fisher,R.A., "The use of multiple measurements in taxonomic problems", Annals of Eugenics, vol.7(II), pp. 179-188, (1936).

[20] Specht, D.F., "Generation of polynomial discriminant functions for pattern recognition", IEEE Transactions on Electron. Comput., vol. 16, pp. 308-319, (1967).

[21] McCulloch, W.S. and Pitts, W.," A logical calculus of the ideas immanent in nervous activity." Bull of Math Biophysics, vol. 5,pp. 115–133,(1943).

[22] Hebb, D., "The organization of behavior.", Wiley, New York, (1949).

[23] Rosenblatt, R., "Principles of neurodynamics". Spartan Books, New York, (1962).

[24] Widrow, B., and Hoff, M.E., "Adaptive switching circuits.", IRE Eastern Electronic Show & Convention (WESCON1960), Convention Record, vol. 4,pp.96–104, (1960).

[25] Grossberg, S., "Neural expectation: Cerebellar and retinal analogues of cells fired by unlearnable and learnable pattern classes." Kybernetik, vol. 10, pp. 49–57, (1972).

[26] Grossberg, S., "Adaptive pattern classification and universal recording: I. Parallel development and coding of neural feature detectors; II. Feedback, expectation, olfaction, and illusions." Biol Cybern, vol. 23, pp. 121–134 & 187–202, (1976).

[27] Fukushima, K., "Cognition: A self-organizing multulayered neural network.", Biol Cybern,vol. 20,pp. 121–136, (1975).

[28] Du K.L. and Swamy M.N.S., " Neural Networks in a Soft-computing Framework. ", Springer-Verlag London Limited ,(2006).

[29] Rumelhart, D.E., Hinton G.E., and Williams R.J., "Learning internal representations by error propagation.", MIT Press, Cambridge, vol. 1,pp. 318–362, (1986).

[30] Werbos, P.J., "Beyond regressions: New tools for prediction and analysis in the behavioral sciences." PhD Thesis, Harvard University, Cambridge, MA, (1974).

[31] Werbos P.J., "Backpropagation through time: What it does and how to do it." Proc. IEEE ,vol. 78(10),pp.:1550–1560, (1990).

[32] Widrow, B., Hoff, M.E.,"Adaptive switching circuits." IRE Eastern Electronic Show & Convention (WESCON1960), Convention Record,vol. 4,pp. 96–104, (1960).

[33] Kohonen, T., "Correlation matrix memories.", IEEE Trans Computers, vol. 21, pp. 353–359, (1972).

[34] MacQueen,J.B., "Some methods for classification and analysis of multivariate observations.", Proc 5th Berkeley Symp on Math Statistics and Probability, University of California Press, Berkeley, pp. 281–297, (1967).

[35] Ackley, D.H., Hinton, G.E. and Sejnowski, T.J., "A learning algorithm for Boltzmann machines." ,Cognitive Sci, vol. 9, pp. 147–169, (1985).

[36] Kirkpatrick, S., Gelatt, J., Vecchi, M.P., "Optimization by simulated annealing. Science." ,vol. 220, pp. 671–680, (1983).

[37] Kohonen, T., "A Simple Paradigm for the Self-Organized Formulation of Structured Maps," Competition and Cooperation in Neural Nets, (Eds.) S. Amari, M. Arbib, Vol. 45, Berlin, Springer-Verlag (1982).

[38] Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". Proceedings of The National Academy Sciences (USA), vol. 79, pp 2554 – 2558, (1982).

[39] Hopfield,J.J, "Neurons with Graded Responses Have Collective Computational Properties Like Those of Two-State Neurons", in Proceedings of The National Academy Sciences (USA), vol. 81, pp. 3088 – 3092, (1984).

[40] Barto, A.G., "Reinforcement learning and adaptive critic methods." , White, D.A., Sofge, D.A., (eds) Handbook of intelligent control: neural, fuzzy, and adaptive approaches, Van Nostrand Reinhold, New York, pp. 469–471, (1992).

[41] Barto, A.G., Sutton, R.S. and Anderson, C.W. ,"Neuronlike adaptive elements that can solve difficult learning control problems.", IEEE Trans Syst Man Cybern,vol. 13,pp. 834–846, (1983).

[42] Kaelbling, L.P., Littman, M.H. and Moore, A.W., "Reinforcement learning: A survey. Journal Artificial Intelligence Research, vol. 4, pp. 237–285, (1996).

[43] Yao, X., "Evolving artificial neural networks.", Proceedings IEEE, vol. 87(9), pp. 1423–1447, (1999).

[44] Kohonen, T., "Self-organized formation of topologically correct feature maps.", Biol Cybern,vol. 43, pp. 59–69, (1982).

[45] Farhat, N.H., Psaltis, D., Prata, A. and Paek, E., "Optical implementation of the Hopfield model.", Appl Optics, vol. 24, pp. 1469–1475, (1985).

[46] Boyd, G.D., "Optically excited synapse for neural networks.", Appl Optics vol. 26, pp. 2712–2719, (1987).

[47] Gadea, R., Cerda, J., Ballester, F. and Mocholi, A., "Artificial neural network implementation on a single FPGA of a pinelined on-line backprogation.", Proc 13th Int Symp Syst Synthesis, Madrid, Spain, pp. 225–230, (2000).

[48] Pineda, F.J., "Generalization of back-propagation to recurrent neural networks.", Physical Rev Letter, vol. 59, pp. 2229–2232, (1987).

[49] Minsky, M.L. and Papert, S., "Perceptrons.", MIT Press, Cambridge, MA, (1969).

[50] Finnoff, W., "Diffusion approximations for the constant learning rate backpropagation algorithm and resistance to local minima.", Neural Comp. , vol. 6(2), pp. 285–295, (1994).

[51] LeCun, Y., Simard, P.Y., and Pearlmutter, B., "Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors.", Hanson SJ, Cowan JD, Giles CL (eds) Advances in neural information processing systems, vol. 5, pp. 156–163, (1993).

[52] Cetin, B.C., Burdick, J.W. and Barhen, J., "Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks. ", Proceedings IEEE International Conference on Neural Networks, San Francisco, pp. 836–842, (1993).

[53] Barhen, J., Protopopescu, V., and Reister, D., "TRUST: A deterministic algorithm for global optimization.", Science, vol. 276,pp. 1094–1097, (1997).

[54] Huber, P.J. , "Robust statistics.", Wiley, New York, (1981).

[55] Chen, D.S. and Jain, R.C., "A robust backpropagation learning algorithm for function approximation.", IEEE Transaction on Neural Networks, vol. 5(3), pp. 467–479, (1994).

[56] Chuang, C.C., Su, S.F. and Hsiao, C.C., "The annealing robust backpropagation (ARBP) learning algorithm.", IEEE Transaction on Neural Networks, vol. 11(5), pp. 1067–1077, (2000).

[57] Zadeh, L.A. "Fuzzy sets. Info & Contr" , vol. 8, pp. 338–353, (1965).

[58] Zadeh, L.A., "The concept of a linguistic variable and its application to approximate reasoning–I, II, III.", Info Sci vol. 8, pp. 199–249, 301–357; 9:43–80, (1975).

[59] Kosko B (1992) Fuzzy system as universal approximators. In: Proc IEEE Int Conf Fuzzy Syst, San Diego, CA, 1153–1162

[60] Mamdani EH (1974) Application of fuzzy algorithms for control of a simple dynamic plant. Proc IEEE 12(1):1585–1588

[61] Takagi T, SugenoM (1985) Fuzzy identification of systems and its applications to modelling and control. IEEE Trans Syst Man Cybern 15(1):116–132

[62] Kosko, B., "Fuzzy engineering.", Prentice Hall, Englewood Cliffs, NJ, (1997).

[63] Yen, J., "Fuzzy logic—A modern perspective." , IEEE Transaction on Knowledge Data Eng, vol. 11(1), pp. 153–165, (1999).

[64] Wang, L.X., "Fuzzy systems are universal approximators.", Proceedings IEEE International Conference on Fuzzy Systems, San Diego,CA, pp. 1163–1170, (1992).

[65] Buckley, J.J., "Sugeno type controllers are universal controllers.", Fuzzy Sets Syst, vol. 53, pp. 299–304, (1993).

[66] Buckley, J.J., Hayashi, Y. and Czogala, E., "On the equivalence of neural nets and fuzzy expert systems.", Fuzzy Sets Syst, vol. 53, pp. 129–134, (1993).

[67] Jin, Y., "Advanced fuzzy systems design and applications.", Physica-Verlag, Heidelberg, (2003).

[68] Figueiredo, M. , Gomides, F., Rocha, A. and Yager, R. , "Comparison of Yager's level set method for fuzzy logic control with Mamdani and Larsen methods.", IEEE Transactions on Fuzzy Systems, vol. 2, pp. 156–159, (1993).

[69] Ramot, D., Friedman, M., Langholz, G. and Kandel, A., "Complex fuzzy logic.", IEEE Transaction on Fuzzy Systems, vol. 11(4), pp. 450–461, (2003).

[70] Ramot, D., Milo, R., Friedman, M. and Kandel, A., "Complex fuzzy sets.", IEEE Transaction on Fuzzy System, vol. 10(2) , pp. 171–186, (2002).

[71] Fogel, D.B., "Evolutionary computation.", IEEE Press, New Jersy, (1995).

[72] Eiben, A.E. and Smith, J.E., "Introduction to Evolutionary Computing.", Springer, ISBN 3-540-40184-9 ,(2003).

[73] Holland, J., "Adaptation in natural and artificial systems.", University of Michigan Press, Ann Arbor, Michigan, (1975).

[74] Rudolph, G., "Convergence analysis of canonical genetic algorithm.", IEEE Transactons of Neural Networks, vol. 5(1), pp. 96–101, (1994).

[75] Koza, John, Forest Bennett, David Andre and Martin Keane. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, (1999).

[76] Smith, J. and Vavak, F., "Replacement strategies in steady state genetic algorithms: Static environments." , Banzhaf W, Reeves C (eds) Foundations of genetic algorithms, Morgan Kaufmann, CA, vol. 5, pp. 219–233, (1999).

[77] Jong, K.D., "An analysis of the behavior of a class of genetic adaptive systems.", PhD Thesis, University of Michigan, Ann Arbor, (1975).

[78] Frantz,D.R., "Non-linearities in genetic adaptive search.", PhD thesis, University of Michigan, Ann Arbor, (1972).

[79] Syswerda, G., "Uniform crossover in genetic algorithms.", Schaffer JD (ed) Proceedings, 3rd International Conference on Genetic Algorithms, Fairfax, VA, pp. 2–9, (1989).

[80] Bhandari, D., Pal, N.R. and Pal, S.K., "Directed mutation in genetic algorithms.", Information science, vol. 79(3/4), pp. 51–270,(1994).

[81] Jerne, N.K., "Towards a network theory of the immune system.", Annales d'Immunologie (Paris), vol. 125(C) ,pp. 373–389, (1974).

[82] Ada,G.L. and Nossal,G.J.V., "The clonal selection theory.", Science Amer vol. 257(2), pp. 50– 57, (1974).

[83] Jiao, L. and Wang, L., "A novel genetic algorithm based on immunity.", IEEE Transaction on System Man Cybern–A, vol. 30(5), pp. 552–561, (2000).

[84] Dorigo, M., Maniezzo, V. and Colorni, A., "Positive feedback as a search strategy." Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, Tech Report, pp. 91-116, (1991).

[85] Dorigo, M. and Gambardella, L.M., " Ant colony system: A cooperative learning approach to the traveling salesman problem.",IEEE Transaction on Evolutionary Computations, vol. 1(1), pp. 53– 66,(1997).

[86] Dorigo, M., Caro, G.D., and Gambardella, L.M., "Ant algorithms for discrete optimization.", Artificial Life vol. 5(2), pp. 137–172, (1999).

[87] Dorigo, M. and Stutzle, T., "Ant colony optimization." MIT Press, Cambridge, MA, (2004).

[88] Montana, D.J. and Davis, L. "Training feedforward networks using genetic algorithms. In: Sridhara N (ed) Proc 11th Int Joint Conf Artif Intell, Detroit, 762–767. Morgan Kaufmann, San Mateo, CA, (1989).

[89] Hinton, G.E. and Nowlan, S.J., "How learning can guide evolution.", Complex Systems, vol. 1, pp. 495–502, (1987).

[90] Whitley, L.D., Gordon V.S. and Mathias, K.E., "Lamarckian evolution, the Baldwin effect and function optimization.", Davidor Y, Schwefel HP, Manner R (eds) Parallel problem solving from nature III, Springer- Verlag, London, UK, (1994).

[91] Turney, P., "Myths and legends of the Baldwin effect.", Proceedings 13th International conference Machine learning, Bari, Italy, 135–142, (1996)

[92] Cooper, M.G. and Vidal, J.J., "Genetic design of fuzzy controllers.", Pal SK, Wang PP (eds) Genetic algorithms for pattern recognition, CRC Press, Boca Raton, FL, pp. 283–298. (1996).

[93] Yip,P.P.C. and Pao, Y.H., "Combinatorial optimization with use of guided evolutionary simulated annealing.", IEEE Transaction on Neural Networks vol. 6(2), pp. 290–295, (1995)

[94] Frean, M., "The upstart algorithm: A method for constructing and training feedforward neural networks.", Neural Computations, vol. 2(2), pp. 198–209, (1990).

[95] Sanger, T.D., "A tree-structured adaptive network for function approximation in high dimensional space.", IEEE Transaction on Neural Networks, vol. 2(2), pp. 285–293, (1991).

[96] Yao, X., "Evolving artificial neural networks.", Proceedings IEEE, vol. 87(9), pp. 1423–1447, (1999).

[97] Lee, S.W., "Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network.", IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 18(6), pp. 648– 652, (1996).

[98] Menczer, F. and Parisi, D., "Evidence of hyperplanes in the genetic learning of neural networks.", Biol Cybern, vol. 66, pp. 283–289, (1992).

[99] Leung, F.H.F., Lam, H.K., Ling, S.H. and Tam, P.K.S., "Tuning of the structure and parameters of a neural network using an improved genetic algorithm.", IEEE Transaction on Neural Networks, vol. 14(1), pp. 79–88, (2003).

[100] Castillo, P.A., Merelo, J.J., Rivas, V., Romero, G. and Prieto, A., "G-Prop: Global optimization of multilayer perceptrons using GAs.", Neurocomputing, vol. 35, pp. 149–163, (2000).

[101] Chalmers, D.J., "The evolution of learning: An experiment in genetic connectionism.", Touretzky DS, Elman JL, Hinton GE (eds) Proc 1990 Connectionist Models Summer School, 81–90. Morgan Kaufmann, San Mateo, CA

[102] Kim, H.B., Jung, S.H., Kim, T.G. and Park, K.H., "Fast learning method for backpropagation neural network by evolutionary adaptation of learning rates.", Neurocomputing, vol. 11(1), pp. 101–106, (1996).

[103] Alvarez, A. , "A neural network with evolutionary neurons.", Neural Process Letters, vol. 16, pp. 43–52, (2002).

CHAPTER 4

Analysis of Pattern Classification for Hand written English Alphabets with soft computing approach

CHAPTER 4: Analysis of Pattern Classification for the Hand Written English Alphabets with Soft Computing Approach.

Abstract

This chapter describes the analysis of pattern classification for hand written English alphabets with feed forward neural network. The neural network with the two hybrid evolutionary algorithm (EAs) is used to analyze the performance for the given task. In this present work we describe the simulation of two hybrid evolutionary algorithms to the feed forward neural network used in classification of the hand written English alphabets. Besides backpropagation algorithm, the soft computing approaches in the form of random genetic algorithm and hybrid evolutionary algorithm have also been considered. The objective is to analyze the performance of soft computing method over other discussed algorithms for the given task. In order to accomplish this task the experiments considered the two different feed forward neural networks trained for the five numbers of trials with hybrid evolutionary algorithm and random genetic algorithm. Each of these algorithms has been taken the definite lead on the conventional approaches of neural network for pattern recognition. It has been analyzed that the feed forward neural network with the genetic algorithm makes better generalization accuracy in character recognition problems. The problem of not convergence the weights in conventional back-propagation has also eliminated by using the soft computing techniques. It has been observed that, there are more then one converge weight matrix in character recognition for every training set. The results of the experiments show that the hybrid evolutionary algorithms can solve challenging problem most reliably and efficiently.

4.1 *Introduction*

Pattern recognition aims to classify data (patterns) based on either a priori knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space. A complete pattern recognition system consists of a sensor that gathers the observations to be classified or described; a feature extraction mechanism that computes numeric or symbolic information from the observations; and a classification or description scheme that does the actual job of classifying or describing observations, relying on the extracted features [1].

Character recognition plays an important role in today's life [2]. It can solve many complex problems in of real life. An example of character recognition is Handwritten English alphabets. The classic difficulty of being able to correctly recognize even typed optical language symbols is the complex irregularity among pictorial representations of the same character due to variations in fonts, styles and size. This irregularity undoubtedly widens when one deals with handwritten characters [3].Classification method designs are based on the following concepts:

Member-roster concept: Under this template-matching concept, a set of patterns belonging to a same pattern is stored in a classification system. When an unknown pattern is given as input, it is compared with existing patterns and placed under the matching pattern class. ***Common property concept***: In this concept, the common properties of patterns are stored in a classification system. When an unknown pattern comes inside, the system checks its extracted common property against the common properties of existing classes and places the pattern/object under a class, which has similar, common properties. ***Clustering concept***: Here, the patterns of the targeted classes are represented in vectors whose components are real numbers. So, using its clustering properties, we can easily classify the unknown pattern. If the target vectors are far apart in geometrical arrangement, it is easy to classify the unknown patterns. If they are nearby or if there is any overlap in the cluster arrangement, we need more complex algorithms to classify the unknown patterns [4-5].

Bayesian decision theory is a system that minimizes the classification error. Bayesian decision theory has a conceptual clarity leading to an elegant numerical recipe. This algorithm can deal with a broader scope of stochastic models than classical algorithms [6]. Nearest neighbor rule [7] is used to classify the handwritten characters. The distance measured between two character images is needed to use this rule. This algorithm works well when the target patterns are far apart. Training in the nearest neighbor rule is very fast. Linear classification or discrimination [5] deals with assigning a new point in a vector space to a class separated by a boundary. It is well suited to mixed data types. It can also handle non-linear cases and missing data. The results produced by the system are very easy to interpret.

All the methods mentioned above have their limitations such as bayesian decision theory has computational difficulties .This mans that the method has a difficulty filling in numerical details. The method also has an obligation to use prior information; if unavailable the theory will not work properly. The basic problem with the nearest neighbor rule is it requires the large set of data and the query time is very slow. Because of the larger set of data it is very much prone to data error. Therefore if there were any irrelevant information entered into the system the system will be easily misinterpret the results. The same problem of larger data set occurs with the linear classification method [8].

All these aforesaid tasks can easily be accomplished by a human being without involving much effort due to its complex structure and working in parallel mechanism. The structure of biological neural network [9] has been simulated and modeled in a serial fashion that provides parallelism through ANN. One of the important advantages of ANN is its adaptive nature [10] and due to this property many existing paradigms can be fused into it easily. The powerful attribute of neural network is the ability to learn arbitrary non-linear mapping using one of the appropriate learning rules. Once the ANN system has trained, it can use for the pattern classification [11], pattern association, pattern mapping, pattern grouping [11], and feature mapping pattern and optimization control etc. Accomplish the task of pattern classification & pattern mapping the supervised multilayer feed forward neural network [12,13] is considered with non-linear differentiable function in all processing units of output and hidden layers. The number of processing units in the input layer, corresponds to the dimensionalities of the input pattern, are linear. The number of output units corresponds to the number of distinct classes in the pattern classification. A method has been developed [14], so that network can be trained to capture the mapping explicitly in the set of input output pattern pair collected during an experiment and simultaneously expected to modal the unknown system for function from which the predictions can be made for the new or untrained set of data. The possible output pattern class would be approximately an interpolated version of the output pattern class corresponding to the input learning pattern close to the given test input pattern. This method involves the back propagation-learning rule [15] based on the principle of gradient descent along the error surface in the weight space. This algorithm is used for the training of a supervised multi-layer feed forward neural network, so that the network could be trained to capture the missing implicit pattern and generate the classification for different features in the given set of input-output pattern pairs.

Efficient learning by the back-propagation (BP) algorithm is required for many practical applications [16]. The BP algorithm calculates the weight changes of artificial neural networks, and a common approach is to use a two-term algorithm consisting of a learning rate (LR) and a momentum factor (MF). The major drawbacks of the two-term BP learning algorithm are the problems of local minima and slow convergence speeds, which limit the scope for real-time applications [17]. A local minimum is defined as a point such that all points in a neighborhood have an error value greater than or equal to the error value in that point. [18].However, GA is particularly good to perform efficient searching in large and complex space to find out the global optima and for the convergence. As the complexity of the search space increases, GA presents an increasingly attractive alternative to gradient based techniques to solve many practical problems [19].

The proposed experiments demonstrate that for the given set of problem (recognition of English hand written alphabets) the performance of hybrid evolutionary feed forward neural network is better than the other algorithm discussed in terms of the accuracy and rate of convergence. We found the significant difference in accuracy and the rate of convergence of hybrid evolutionary algorithm (hybrid genetic algorithm) with the simple back propagation algorithm and random genetic algorithm (random search algorithm) for the given set of the problem.

Section two of this chapter describes the generalized approaches and principals of the algorithms applied for the problem. Section three describes the architecture and design of the network used, in terms of simulation design. Section four shows the results of the experiments. Section five describes the discussions and the future work based on the results, while in the last section conclusion and summary of the paper is presented.

4.2 Supervised Feed Forward Neural Network

Feed forward neural network is a biologically inspired classification algorithm. It consists of a number of simple neuron-like processing *units*, organized in *layers*. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal; each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called *nodes*. Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called *feed forward* neural networks. The neural network consists of an input layer of nodes, one or more hidden layers, and an output layer. Each node in the layer has one corresponding node in the next layer, thus creating the stacking effect. The input layer's nodes have output functions that deliver data to the first hidden layer nodes. The hidden layer(s) are the processing layer, where all of the actual computation takes place. Each node in a hidden layer computes a sum based on its input from the previous layer (either the input layer or another hidden layer). The sum is then "compacted" by a sigmoid function (a logistic curve), which changes the sum to a limited and manageable range. The output sum from the hidden layers is passed on to the output layer, which produces the final network result as shown in figure 4.1. Feed-forward networks may contain any number of hidden layers, network with a single hidden layer can learn any set of training data that a network with multiple layers can learn, depends upon the complexity of the problem [20]. However, neural network with a single hidden layer may take longer to train.

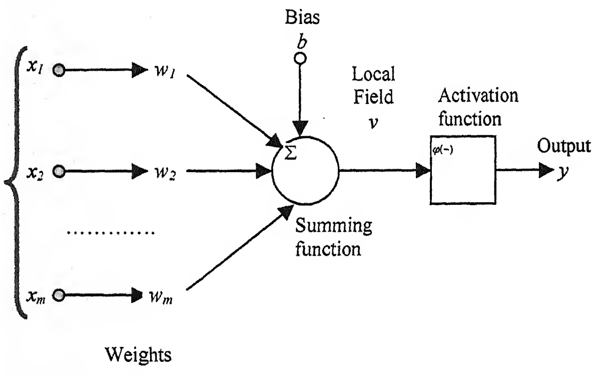


Figure 4.1: The functioning of neural network architecture.

An input may be either a raw / preprocessed signal or image. Alternatively, some specific features can also be used. If specific features are used as input, their number and selection is crucial and application dependent. Weights are connected between an input and a summing node. These affect to the summing operation. The quality of network can be seen from weights Bias is a constant input with certain weight. Usually the weights are randomized in the beginning [21,22].

The neuron is the basic information processing unit of a NN. It consists of: A set of links, describing the neuron inputs, with weights, $w_1\ w_2\ , w_3 \ldots\ldots w_m$, An adder function (linear combiner) for computing the weighted sum as:

$$v = \sum_{j=1}^{m} w_j x_j \tag{4.1}$$

and activation function (squashing function) for limiting the amplitude of the neuron output as shown in figure 4.1.

$$y = \varphi(v + b) \tag{4.2}$$

where,

$$v = \sum_{j=0}^{m} w_j x_j \tag{4.3}$$

$$b = w_0$$

The output at every node can finally calculates by using sigmoid function

$$y = f(x) = \frac{1}{1 + e^{-x}} \quad ; \tag{4.4}$$

where k = 1 (constant)

The Feed Forward Neural Network uses a *supervised* learning algorithm: besides the input pattern, the neural net also needs to know to what category the pattern belongs. Learning proceeds as follows: a pattern is presented at the inputs. The pattern will be transformed in its passage through the layers of the network until it reaches the output layer. The units in the output layer all belong to a different category. The outputs of the network as they are now compared with the outputs as they ideally would have been if this pattern were correctly classified, in the latter case the unit with the correct category would have had the largest output value and the output values of the other output units would have been very small. On the basis of this comparison all the connection weights are modified a little bit to guarantee that, the next time this same pattern is presented at the inputs, the value of the output unit that corresponds with the correct category is a little bit higher than it is now and that, at the same time, the output values of all the other incorrect outputs are a little bit lower than they are now. The differences between the actual outputs and the idealized outputs are propagated back from the top layer to lower layers to be used at these layers to modify connection weights. This is why the term back-*propagation network* is also often used to describe this type of neural network.

A genetic algorithm (GA) [23] is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a *fitness function* that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random.

The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem. GA differs from more traditional optimization techniques in that they involve a search from a "population" of solutions, not from a single point. Each iterations of GA involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen [19, 20].

A population of individuals is maintained within **search space** for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the **binary** alphabet **{0, 1}**. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A **fitness score** is assigned to each solution representing the abilities of an individual to `compete'. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective `breeding' of the solutions to produce `offspring' better than the parents by combining information from the chromosomes.

The GA maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced with a new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted. In this way it is hoped that over successive generations better solutions will thrive while the least fit solutions die out. New generations of solutions are produced containing, on average, better genes than a typical solution in a previous generation. Each successive generation will contain more good `partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

After an initial population is randomly generated, the algorithm evolves through the three operators:

1. **selection** which equates to survival of the fittest;
2. **crossover** which represents mating between individuals.
3. **mutation** which introduces random modifications.

Selection Operator gives preference to better individuals, allowing them to pass on their genes to the next generation. The goodness of each individual depends on its fitness. Fitness may be determined by an objective function or by a subjective judgment.

Crossover Operator is a prime distinguished factor of GA from other optimization techniques. Two individuals are chosen from the population using the selection operator. A crossover site along the bit strings is randomly chosen and the values of the two strings are exchanged as if S1=000000 and S2=111111 and the crossover point is 2 then S1=110000 and S2=001111.The two new offspring created from this mating are put into the next generation of the population .By recombining portions of good individuals, this process is likely to create even better individuals.

Mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code. With some low probability, of a portion of the new individuals will have some of their bits flipped. Its purpose is to maintain diversity within the population and inhibit premature convergence. Mutation alone induces a random walk through the search space Mutation and selection (without crossover) creates a parallel, noise-tolerant, hill-climbing algorithms.

Popular working –out algorithms for feed forward neural networks such as back-propagation algorithm undergo the intrinsic complications of gradient –decent techniques-predominantly local minima in the error function. GA propose an another solution to conventional techniques since they do not rely on gradient information –they can sample the search space irrespective of where the existing solution is to be found , while remaining is biased toward good solutions.

Lots of work has been already done on the evolution of neural network with hybrid GA [19, 20].The majority of implementations of the GAs are derivatives of Holland's[24] innovative specification. Evolution has been introduced in NNs at three levels: connection weights, architectures and learning rules. The evolution rules have not yet been subjected to a similar study, but the literature on the subject is mounting. The evolution of a network's connection weights is an area of curiosity and the center of attention of this manuscript. The GA is used in the feed forward neural network for evolving the population of the weights is evaluated from the fitness evaluation function.

The least mean square error function is used for the evaluation of individual weight population. The fittest weights are used for further computation and participate in the classification. This process will continue until the required classification is achieved. The final selected weights represent the optimized strength of the connection in the network architecture, which is suitable for the classification of all presented patterns. This strength represents the convergence of the weights for the desired classification. There is also a possibility that more than one population of weights is generating the correct classification for every presented training pattern. [8, 9].

4.3 Simulation Design and Implementation Details

We have considered two different architecture first consisting 4 neurons in the input layer, 6 neurons in each hidden layers, 2 hidden layers and 5 neurons in the output layer, while the second architecture consists 4 neurons in the input layer, 5 neurons in each hidden layers, 2 hidden layers and 5 neurons in the output layer.

4.3.1 Experiment

Five different sets of alphabets are used for the both experiment. The alphabets are converted into their density function by using MATLAB program, for input data. We used three different learning algorithms: backpropagation algorithm, random genetic algorithm(hybrid random search algorithm), and hybrid evolutionary algorithm(hybrid genetic algorithm). In the each experiment we have taken five trials to train the neural network population through three different algorithms.

The genetic operators used in each experiment are summarized in Table 4.1:

Table 4.1: Genetic operators used in the experiments

Training Algorithms	**Genetic Operators Used**
Back-propagation	None
GA	Mutation with probability <= 0.1 and Crossover
Hybrid EA	Mutation with probability <= 0.1 and Crossover

The parameters used in all three experiments are listed in Table 4..2 and 4..3.

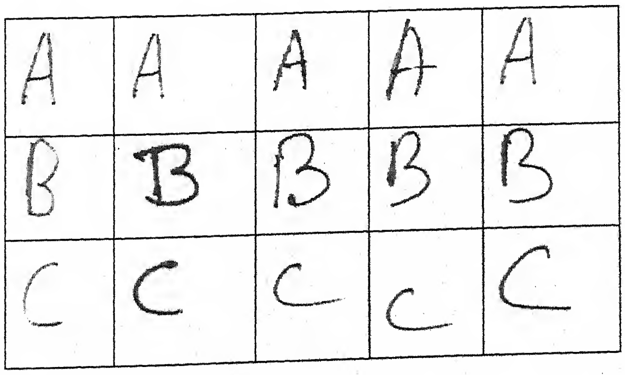
Table 4.2: Parameters used for Back propagation Algorithm

Parameter	**Value**
Back-propagation learning Rate (η)	0.1
Momentum Term (α)	0.9
Doug's Momentum Term $\left(\frac{1}{1-(\alpha)}\right)$	$\left(\frac{1}{1-(\alpha)}\right)$
Adaptation Rate (K)	3.0
Minimum Error Exist in the Network $(MAXE)$	0.00001
Initial weights and biased term values	Randomly Generated Values Between 0 and 1

Table 4.3: Parameters used for Genetic Algorithm and Hybrid Evolutionary Algorithm

Parameter	**Value**
Adaptation Rate (K)	3.0
Back-propagation learning Rate (η)	0.1
Momentum Descent Term (α)	0.9
Doug's Momentum Term $\left(\frac{1}{1-(\alpha)}\right)$	$\left(\frac{1}{1-(\alpha)}\right)$
Mutation Population Size	3
Crossover Population Size	2000
Minimum Error Exist in the Network $(MAXE)$	0.00001
Initial weights and biased term values	Randomly Generated Values Between 0 and 1

In all experiments, the neural networks were trained to generate the appropriate classification for the handwritten English alphabets. For this, the scanned images of five different samples of handwritten English alphabets (Fig. 4.2) were obtained.



D	D	D	D	D
E	E	E	E	E
F	F	f	f	F
G	G	G	G	G
H	H	H	H	H
I	I	I	I	I
J	J	J	J	J
K	K	K	K	K
L	L	L	L	L
M	M	M	M	M
N	N	N	N	N
O	O	O	O	O
P	P	P	P	P
Q	Q	Q	q	Q
R	R	R	R	R

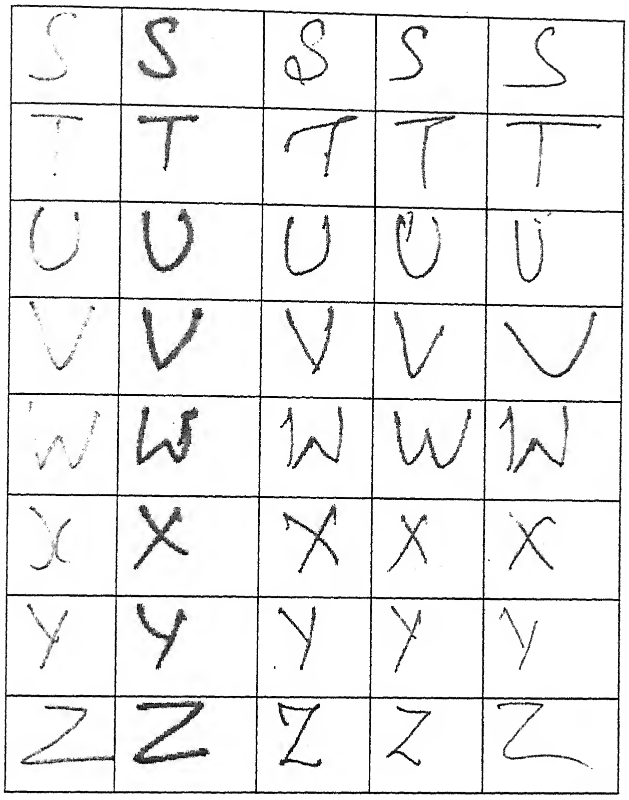


Figure 4.2: Scanned images of five different samples of handwritten English alphabets

All collected hand written alphabets images were partitioned into four equal parts, and the density values of the pixels for each part were calculated. Next, the density values of the central of gravities for these partitioned images were calculated. Consequently four values were obtained from an image of handwritten English alphabets, which were then used as the input for the feed-forward neural network. This procedure was used to present the input pattern to the feed-forward neural network for each of the sample of English Alphabets.

4.3.2 The Neural Network Architecture

The architecture of the neural networks was based on a fully connected feed-forward multilayer generalized perceptron. The hidden layers were used to investigate the effects of the algorithms on the hyper plane. Each network had a single output unit with the following activation and output functions,

$$A_k^o = \sum_{i=0}^{H} w_{io_k} O_i^h \tag{4.5}$$

$$\int\left(A_k^o\right) = \int\left(\sum_{i=0}^{H} w_{io_k} O_i^h\right) = O_k^o \tag{4.6}$$

where function $\int\left(A_k^o\right)$ is given as,

$$O_k^o = \frac{1}{1+e^{-KA_k^o}} \tag{4.7}$$

Now, similarly, the output and activation value for the neurons of hidden layers and input layer can be written as,

$$A_k^h = \sum_{i=0}^{N} w_{ik} O_i \tag{4.8}$$

and $$O_k^h = \frac{1}{1+e^{-KA_k^h}} \tag{4.9}$$

and $$O_k^i = \int\left(A_k^i\right) = A_k^i \tag{4.10}$$

In the Back-propagation learning algorithm, the change in weight populations was done according to the calculated error in the network after each of the iteration of training. The change in weight and error in the network can be calculated as follows,

$$\Delta w_{ho}(s+1) = -\eta \sum_{i=1}^{H} \frac{\partial E}{\partial w_{ho}} + \alpha \Delta w_{ho}(s) + \frac{1}{1-(\alpha \Delta w_{ho}(s))} \tag{4.11}$$

$$\Delta w_{ih}(s+1) = -\eta \sum_{i=1}^{N} \frac{\partial E}{\partial w_{ih}} + \alpha \Delta w_{ih}(s) + \frac{1}{1-(\alpha \Delta w_{ho}(s))} \tag{4.12}$$

$$E = \frac{1}{2} \sum_{p=1}^{P} \left(O^{O} - T\right)^{2} \tag{4.13}$$

where $\left(O^{O} - T\right)^{2}$ is the squared difference between the actual output value of the output layer for pattern p and the target output value. Doug's momentum term [3] was used with momentum descent term for calculating the change in weights in equations 4.11 and 4.12. Doug's momentum descent is similar to standard momentum descent with the exception that the pre-momentum weight step vector is bounded so that its length cannot exceed 1.0. After the momentum is added, the length of the resulting weight change vector can grow as high as 1 / (1 - momentum). This change allows stable behavior with much higher initial learning rates, resulting in less need to adjust the learning rate as the training progresses. The evolutionary algorithms evolve the population of weights using its operators, and select the best population of the weights that minimize the error between the desired output and the actual output of neural network system.

4.4 The Genetic Algorithm Implementation

Figure 4.3 shows the standard form of a genetic algorithm. The initial population was generated with randomly assigned values for weights and biases. The values were obtained from the random generator generating values between 0 and 1.

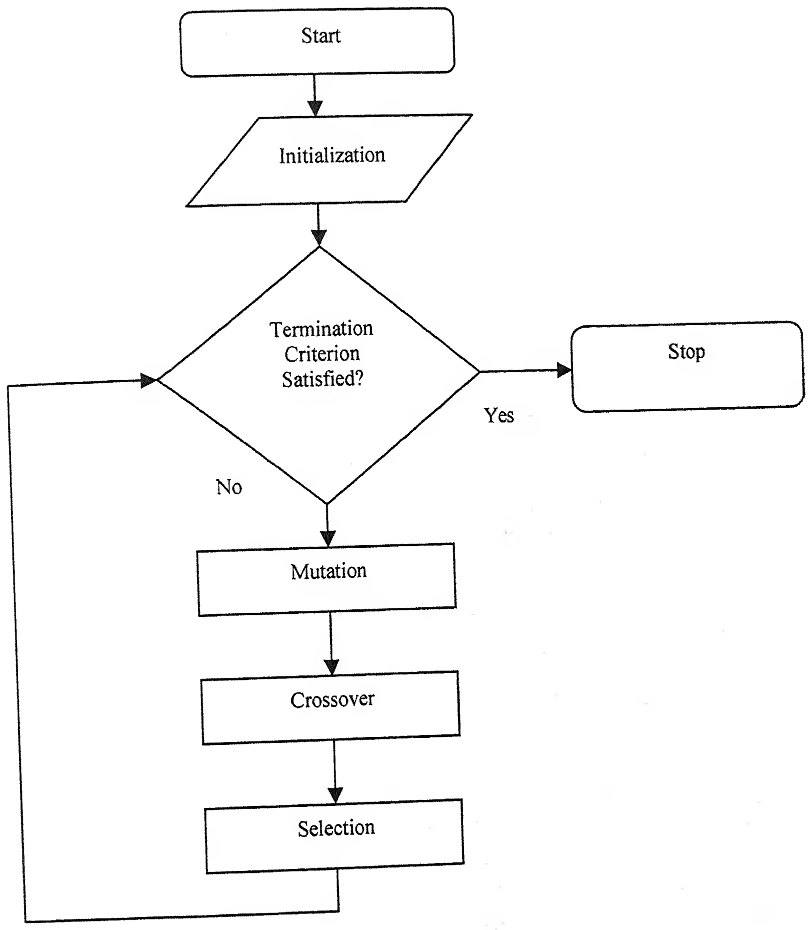


Figure 4.3: Flowchart of Genetic Algorithm Implementation

4.4.1 Genetic Representation

After the initial population of weights and biases was created, the problem domain is represented as a chromosome. For that, the set of weight values is represented as a matrix (Table4.4).

Table 4.4(a): Weight matrix for neural network [4-6-6-5].

To/From	I_1	I_2	I_3	I_4	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{21}	H_{22}	H_{23}	H_{24}	H_{25}	H_{26}
I_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I_4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
H_{11}	W_{11}^{IH}	W_{12}^{IH}	W_{13}^{IH}	W_{41}^{IH}	-	-	-	-	-	-	-	-	-	-	-	-
H_{12}	W_{21}^{IH}	W_{22}^{IH}	W_{32}^{IH}	W_{42}^{IH}	-	-	-	-	-	-	-	-	-	-	-	-
H_{13}	W_{31}^{IH}	W_{23}^{IH}	W_{33}^{IH}	W_{43}^{IH}	-	-	-	-	-	-	-	-	-	-	-	-
H_{14}	W_{14}^{IH}	W_{24}^{IH}	W_{34}^{IH}	W_{44}^{IH}	-	-	-	-	-	-	-	-	-	-	-	-
H_{15}	W_{15}^{IH}	W_{25}^{IH}	W_{35}^{IH}	W_{45}^{IH}	-	-	-	-	-	-	-	-	-	-		
H_{16}	W_{16}^{IH}	W_{26}^{IH}	W_{36}^{IH}	W_{46}^{IH}	-	-	-	-	-	-	-	-	-	-	-	-
H_{21}	-	-	-	-	W_{11}^{HH}	W_{21}^{HH}	W_{31}^{HH}	W_{41}^{HH}	W_{51}^{HH}	W_{61}^{HH}	-	-	-	-	-	-
H_{22}	-	-	-	-	W_{12}^{HH}	W_{22}^{HH}	W_{32}^{HH}	W_{42}^{HH}	W_{52}^{HH}	W_{62}^{HH}	-	-	-	-	-	-
H_{23}	-	-	-	-	W_{13}^{HH}	W_{23}^{HH}	W_{33}^{HH}	W_{43}^{HH}	W_{53}^{HH}	W_{63}^{HH}	-	-	-	-	-	-

H$_{24}$	-	-	-	-	W_{14}^{HH}	W_{24}^{HH}	W_{34}^{HH}	W_{44}^{HH}	W_{54}^{HH}	W_{64}^{HH}	-	-	-	-	-	-
H$_{25}$	-	-	-	-	W_{15}^{HH}	W_{25}^{HH}	W_{35}^{HH}	W_{45}^{HH}	W_{55}^{HH}	W_{65}^{HH}	-	-	-	-	-	-
H$_{26}$	-	-	-	-	W_{16}^{HH}	W_{26}^{HH}	W_{36}^{HH}	W_{46}^{HH}	W_{56}^{HH}	W_{66}^{HH}	-	-	-	-	-	-
O$_1$	-	-	-	-	-	-	-	-	-	-	W_{11}^{HO}	W_{21}^{H0}	W_{31}^{HO}	W_{41}^{HO}	W_{51}^{HO}	W_{61}^{H0}
O$_2$	-	-	-	-	-	-	-	-	-	-	W_{12}^{HO}	W_{22}^{HO}	W_{32}^{HO}	W_{42}^{HO}	W_{52}^{HO}	W_{62}^{HO}
O$_3$	-	-	-	-	-	-	-	-	-	-	W_{13}^{HO}	W_{23}^{HO}	W_{33}^{HO}	W_{43}^{HO}	W_{53}^{HO}	W_{63}^{HO}
O$_4$	-	-	-	-	-	-	-	-	-	-	W_{14}^{HO}	W_{24}^{HO}	W_{34}^{HO}	W_{44}^{HO}	W_{54}^{HO}	W_{64}^{HO}
O$_5$	-	-	-	-	-	-	-	-	-	-	W_{15}^{HO}	W_{25}^{HO}	W_{35}^{H0}	W_{45}^{HO}	W_{55}^{HO}	W_{65}^{HO}

Table 4.4(b): Bias term Matrix for neural network [4-6-6-5].

To/From	H$_{11}$	H$_{12}$	H$_{13}$	H$_{14}$	H$_{15}$	H$_{16}$	H$_{21}$	H$_{22}$	H$_{23}$	H$_{24}$	H$_{25}$	H$_{26}$	O$_1$	O$_2$	O$_3$	O$_4$	O$_5$
Bias	θ_1^{H1}	θ_2^{H1}	θ_3^{H1}	θ_4^{H1}	θ_5^{H1}	θ_6^{H1}	θ_1^{H2}	θ_2^{H2}	θ_3^{H2}	θ_4^{H2}	θ_5^{H2}	θ_5^{H2}	θ_1^{O}	θ_2^{O}	θ_3^{O}	θ_4^{O}	θ_5^{O}

Each row of this matrix represents a group of incoming weighted links to a single neuron. In total, there are 90 weighted links between neurons and 17 biased values of neurons in the neural network [4-6-6-5] while in the network [4-5-5-5] there are 70 weighted links between neurons and 15 biased values of neurons. Thus, a chromosome is a collection of genes representing either a weight or a biased value. Where some gene corresponds to a single weighted link and some corresponds to biased values of neurons in the network.

4.4.2 The Mutation Operator

A mutation operator randomly selects a gene in a chromosome and adds a small random value between –1 and 1 to that particular gene will produce next generation population of 107-gene chromosomes of [4-6-6-5] network and 85-gene chromosomes of [4-5-5-5] network. The size of next generated population will be of size $n+1$ (to meet the elitism), if the mutation operator has applied n times over the old chromosome. Then we have,

$$C^{new} = C^{old} \bigcup_{i=1}^{n} \left[C_{121-\lambda} \bigcup \left(C_{\lambda}^{old} + \in \right) \right] \tag{4.14}$$

where C^{old} symbolizes the old chromosome of 107-gene or 85-gene, $\in$ symbolizes the small randomly generated value between –1 to 1, λ symbolizes the randomly selected gene of C^{old} chromosome for adding the $\in$ and C^{new} symbolizes the next generation population of chromosome, i.e. $C^{new} = [C_1, C_2, C_3, -------, C_n, C_{n+1}]$. The inner

$\bigcup$ operator prepares a new chromosome at each iteration of mutation and outer $\bigcup$ operator is building the new population of chromosomes called C^{new}.

4.4.3 Elitism

Elitism was used when creating each generation so that the genetic operators did not lose good solutions. This involved copying the best-encoded network unchanged into the new population , which includes C^{old} for creating .

4.4.4 Selection

This will select a chromosome C^{sel} among the mutated population of chromosomes for which the sum of squared errors is minimum for the feedforward neural network, i.e. iteratively all the chromosomes values will be assigned to the network architecture in terms of weights and biased values defined in chromosome. After assigning the values, the each network architecture, will be able to fabricate output using these assigned values. For each chromosome of C^{new}, the error can be calculated using these fabricated outputs. Now, the selection operator will pick a chromosome C^{sel} from C^{new}, which generates minimized error for the network.

4.4.5 Crossover

The crossover operator takes selected chromosome and creates a child for producing the next generation population of 107-gene or 85-gene chromosomes of size $n+1$. This next generation of population is getting by applying the crossover operator n times. Experiments performed the crossover by Swapping the two randomly selected gene values of the parent chromosome as given in figure 4.4(a), using,

$$C^{next} = C^{sel} \bigcup_{i=1}^{n} \left[\left(C^{sel} - C_{\alpha}^{sel} - C_{\beta}^{sel} \right) \bigcup \left(C_{\alpha}^{sel} \xleftrightarrow[v_{\alpha} \leftrightarrow v_{\beta}]{} C_{\beta}^{sel} \right) \right] \tag{4.15}$$

where α and β symbolize the randomly generated genes positions in CP_1 and CP_2 in C^{sel} chromosome and C^{next} is the next generation of size $n+1$.

Chromosome C^{sel}

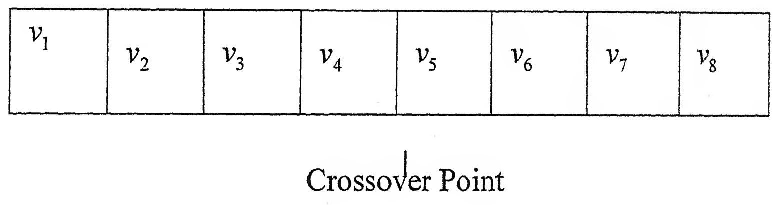


Figure 4.4(A)

Chromosome

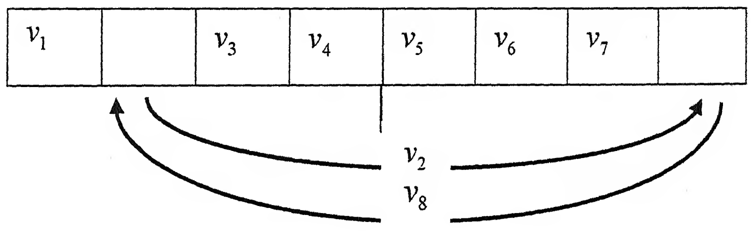


Figure 4.4(B)

Chromosome

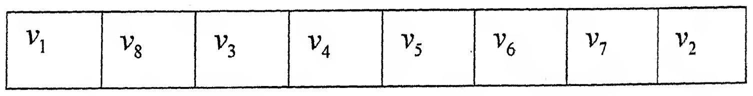


Figure 4.4(C)

Figure 4.4: (A) Chromosome before applying crossover operator, (B) Applying crossover operator on chromosome, and (C) Chromosome after applying crossover operator

4.4.6 Fitness Evaluation Function

This is to define a fitness function for evaluating the chromosome's performance. This function must estimate the performance of the weight population for a given feedforward neural network. We apply here a practically simple function defined by the proportional of the sum of squared errors. To evaluate the fitness of a given chromosome, each weight and biased value contained in the chromosome is assigned to the respective link and neuron in the network. The training set is then presented to the network, and the sum of squared errors is calculated. The smaller the sum, the fitter the chromosome. In other words, the genetic algorithm attempts to find a set of weights and biased values that minimizes the sum of squared errors as,

```
        $\min error = 1.0$
    For all the n+1 chromosomes
        if $(\min error > E_{C_i^{next}})$ then
            $(\min error = E_{C_i^{next}})$
            $C^{\min} = C_i^{next}$
        else
            $(\min error = \min error)$                    (4.16)
```

where $E_{C_i^{next}}$ symbolizes the error calculated for ith chromosome among the $n+1$ chromosomes of C^{next} population, $C^{\min}$ symbolizes the chromosome, which has minimized error.

4.5 Results and Discussion

The results shown below, consist 22 tables (from Table 4.5 to Table 4.26) having entries for iterations and count of convergence weight matrix and 22 graphs (from Figure 4.5 to Figure 4.26). Tables 4.5 [a, b], 4.7 [a, b], 4.9 [a, b], 4.11 [a, b], 4.13 [a, b], 4.15 [a, b], 4.17 [a, b], 4.19 [a, b], 4.21[a, b], 4.23[a, b], contains the integer value for number of iterations performed by each algorithms to recognize the five samples of each hand written English alphabets, while the real values in the tables show that the error exists upto 50000 iterations , means algorithm could not recognize the given sample of alphabets . Tables 4.6 [a, b], 4.8 [a, b], 4.10 [a, b], 4.12 [a, b], 4.14 [a, b], 4.16 [a, b], 4.18 [a, b], 4.20 [a, b], 4.22[a, b], 4.24[a, b] contains the value for number of convergence weight matrix obtained to recognize the given samples by each algorithm, the entry for the count of convergence weight matrix has not shown for back-propagation algorithm because it could not converse the final results in most of the cases. The graphs [Figure 4.5 to 4.24] show the comparisons of the performance of different algorithms based on the results obtained. The network mentioned as first network consisting one input layer with 4 neurons, two hidden layers with 5 neurons and one output layer with 5 neurons[4-5-5-5] while the other network mentioned as second network consisting one input layer with 4 neurons, two hidden layers with 6 neurons and one output layer with 5 neurons[4-6-6-5]. The graphs [Figure 4.5, 4.7,4.9,4.11,4.13,4.15,4.17,4.19,4.21,4.23] are plotted between the average iterations of five samples of any alphabet performed by the different algorithms for each trials, the value 50001 has been taken for iteration where the algorithm could not converged the solution for any sample. While the graphs [Figure 4.6, 4.8, 4.10, 4.12, 4.14, 4.16, 4.18, 4.20, 4.22, 4.24] plotted between the average number of convergence weight matrix for five samples of alphabets.

Table 4.5(a): Results of first trial of first network [4-5-5-5]

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated by Back-Propagation Algorithm													
Sample 1	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated by Genetic Algorithm													
Sample 1	884	953	2	42	1694	3	5	12	74	57	262	514	441
Sample 2	1	2	1	1	2	1	1	1	1	2	2	2	2
Sample 3	1	2	469	1	2	1768	1	1	72	2	2	56	68
Sample 4	1	2	1	1	2	2	1	1	4	1	5	864	1
Sample 5	1	2	1	1	2	10	26	1	3	1	1	1	1
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	54	488	24	2428	1	0.1	0.3	0.1	0.2	0.3	0.3	0.2	0.2
Sample 2	1	1	120	1	1	0.1	0.2	0.2	0.2	0.3	0.3	0.2	0.2
Sample 3	1	1	1	1	1	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2
Sample 4	1	1	1	1	1	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2
Sample 5	1	1	1	1	1	0.2	0.2	0.2	0.3	0.2	0.3	0.2	0.2

Table 4.5(b): Results of first trial for first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated by Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated by Genetic Algorithm													
Sample 1	175	1	2806	54	16	161	6	11	865	35	210	818	292
Sample 2	1	1	609	2	1	2	313	1821	2	2	123	112	2
Sample 3	1	1	2	1	1	35	1	4	2	193	2	2	1166
Sample 4	1	1	2	1	1	1	1	2	420	2	4	1	2
Sample 5	1	1	1	1	1	1	1	18	4	2	111	30	2
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.3	0.1	0.2	0.3	0.3	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2

Figure 4.5: The Comparison Chart for iterations for recognition of handwritten English alphabets for first trial of simulation for network [4-5-5-5].

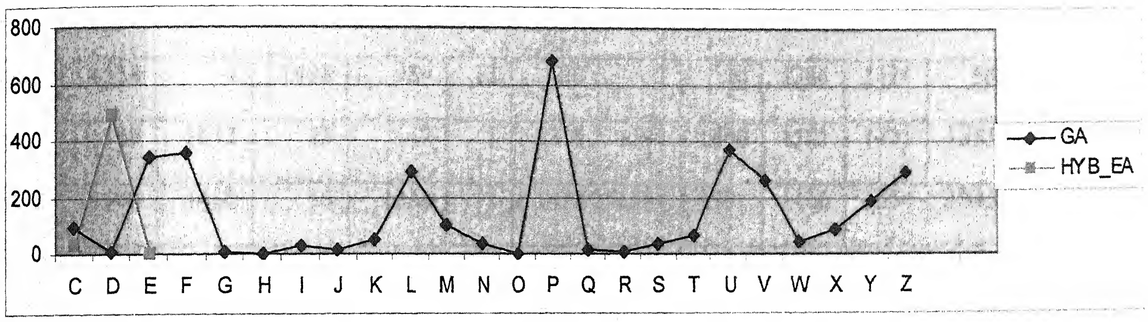


Table 4.6(a): Results of first trial for first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1220	1130	1	1115	281	3	53	11	1477	1326	1582	1370	1682
Sample 2	1026	1094	1	1122	1477	7	60	12	1535	146	1529	1349	1687
Sample 3	994	1162	1300	1127	317	1499	202	10	1422	1197	1565	168	1540
Sample 4	172	1063	1152	1079	103	1542	1446	7	1473	1157	1572	1321	1501
Sample 5	6	857	1177	1104	1083	1494	110	7	1574	1159	1604	1344	1330
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1739	1823	1785	1882	5	The algorithm could not converge to the solution after alphabet 'E' so no convergence weight matrix obtained for remaining alphabets.							
Sample 2	1698	1888	1839	1895	1733								
Sample 3	1692	141	1757	1904	1731								
Sample 4	1674	1875	1767	1891	1734								
Sample 5	1687	1879	1733	1892	1708								

Table 4.6(b): Results of first trial of for first network [4-5-5-5].

lphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	Number of convergence weight matrix for Genetic Algorithm												
Sample 1	1558	4	1408	989	12	20	7	10	1286	1434	58	1505	1473
Sample 2	189	1612	25	147	7	1318	1394	1600	1301	1421	1355	1639	1438
Sample 3	1546	1460	16	1334	11	1478	1301	1591	1466	1554	1345	1727	1410
Sample 4	1402	1534	17	1098	9	1311	1443	1580	1511	1547	1380	1491	1417
Sample 5	1498	1430	1390	1310	11	1468	1349	1748	1546	1597	997	1421	1248
	Number of convergence weight matrix for Hybrid Evolutionary Algorithm												
Sample 1	The algorithm could not converge to the solution after alphabet 'E' so no convergence weight matrix obtained for remaining alphabets.												
Sample 2													
Sample 3													
Sample 4													
Sample 5													

Figure 4.6: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for first trial of simulation for network [4-5-5-5].

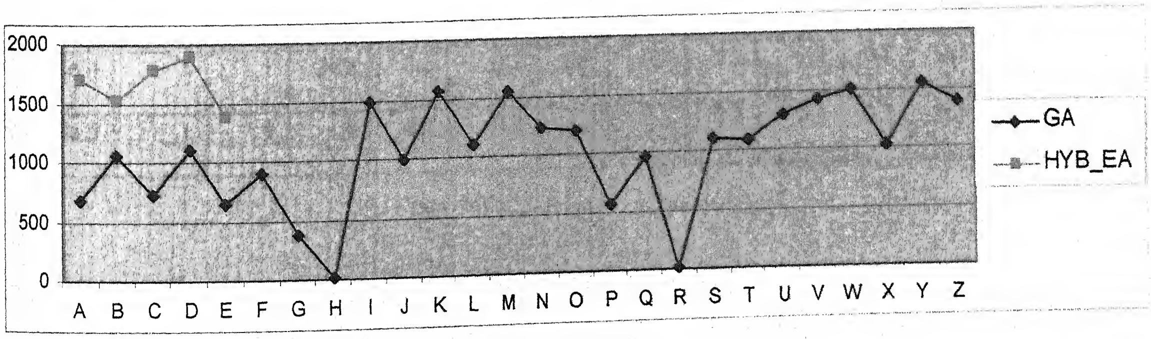


Table 4.7(a): Results of second trial of first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	0.1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	891	12	198	4	368	5	3	46	1091	51	538	1182	2
Sample 2	1	1	1	1	2	1	1	1	47	1	2	1	2
Sample 3	1	1	1	1	2	1	1	1	2	1	2	1	7
Sample 4	1	1	1	1	2	1	1	1	2	1	2	1	1
Sample 5	1	81	1	1	2418	1	195	1	8	4	1	1	4
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	25	670	928	378	0.3	0.1	0.3	0.1	0.2	0.3	0.3	0.2	0.2
Sample 2	1	1	2	0.1	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2	0.2
Sample 3	1	1	90	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2
Sample 4	1	1	209	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2
Sample 5	1	1	1	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2	0.2

Table 4.7(b): Results of second trial of first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	2	252	2274	572	708	399	6	1313	1	1126	163	306	320
Sample 2	1697	1	159	1	1	1	1	2	1	2	2	1	2
Sample 3	23	2	2	20	1	1	6	2	1	2	990	1	3
Sample 4	12	124	1	1	1	1	1	2	1	2	90	1	2
Sample 5	2	1	1	1	1	1	11	2	1	2	1	1	2
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.3	0.1	0.2	0.3	0.3	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2

Figure 4.7: The Comparison Chart for iterations for recognition of handwritten English alphabets for second trial of simulation for network [4-5-5-5].

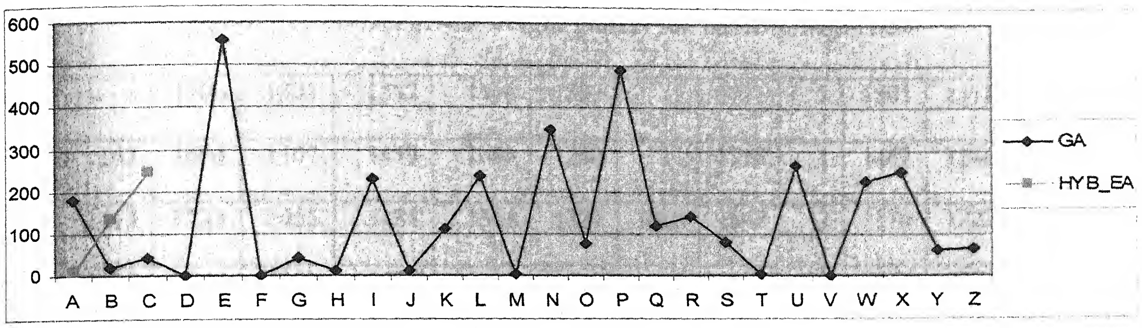


Table 4.8(a): Results second trial of for first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1383	9	891	1	17	1	11	1328	1479	1426	1480	1417	1336
Sample 2	1121	7	1042	4	1201	3	1445	1201	1405	1492	1384	1094	1366
Sample 3	1063	4	982	10	1271	10	1238	1251	1520	1403	125	1256	1099
Sample 4	1238	10	989	8	1344	7	1365	1167	1463	1514	1352	1294	1238
Sample 5	25	889	960	9	1353	8	1377	1335	1428	1469	1192	1308	43
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	4	1845	1601	1726	The algorithm could not converge to the solution after alphabet 'D' so no convergence weight matrix obtained for remaining alphabets.								
Sample 2	176	1833	1735										
Sample 3	1579	1832	1586										
Sample 4	43	1823	1595										
Sample 5	45	1893	1621										

Table 4.8(b): Results second trial of for first network [4-5-5-5].

phabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1	1706	1331	1582	1466	1392	5	1721	1	1503	1518	1420	207
Sample 2	143	1694	1107	1479	1096	1240	11	1718	1	1467	1546	1418	513
Sample 3	1413	1733	1065	1488	1341	1377	1026	1647	3	1376	1222	1492	1467
Sample 4	1775	1677	1382	1620	1172	1167	161	1691	13	1360	1371	1542	1524
Sample 5	1818	1727	1231	1433	1364	1368	1471	1704	10	1537	1205	1537	1518
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	The algorithm could not converge to the solution after alphabet 'D' so no convergence weight matrix obtained for remaining alphabets.												
Sample 2													
Sample 3													
Sample 4													
Sample 5													

Figure 4.8: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for second trial of simulation for network [4-5-5-5].

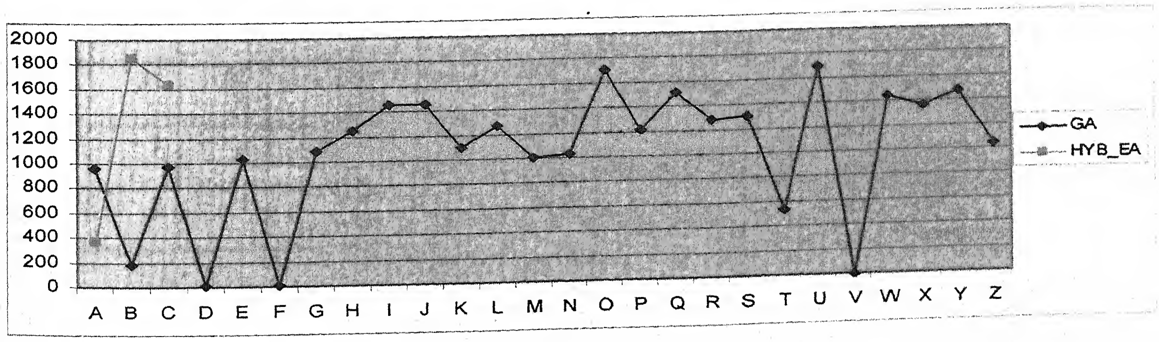


Table 4.9(a): Results of third trial of first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	0.4	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	2061	1186	38	1161	579	31	196	78	2883	912	86	538	1
Sample 2	1	1	1158	2	71	2	1	1	118	1	978	3095	3
Sample 3	1	1	2	2	49	2	153	2	1616	1	3236	1	2
Sample 4	1	1	2	2	2	3535	2	12	2	1	31	1	1
Sample 5	1	1	2	1	2	21	2	1	4	1	1	3	1
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	45	16	786	11	1	464	762	111	0.2	0.3	0.3	0.2	0.2
Sample 2	2	10	1	1	32	1	1	1	0.2	0.3	0.3	0.2	0.2
Sample 3	31	37	1	1	1	1	1	1	0.2	0.2	0.3	0.2	0.2
Sample 4	1	1	1	1	1	1	1	3	0.2	0.2	0.3	0.2	0.2
Sample 5	2	1	1	1	1	1	1	1	0.3	0.2	0.3	0.2	0.2

Table 4.9(b): Results of third trial for first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	2440	580	1219	255	6	1	993	776	398	1	221	122	3
Sample 2	11	2	31	2	73	2	2	1	11	2	1	2	23
Sample 3	2	644	2	5419	1	1	2	1	4	1	1	2	417
Sample 4	2	2	180	4	1	1	2	1	9	1	1	2	2
Sample 5	2	2	2	2	1	1	12749	1	1	1	1	4	2
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.3	0.1	0.2	0.3	0.3	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2

Figure 4.9: The Comparison Chart for iterations for recognition of handwritten English alphabets for third trial of simulation for network [4-5-5-5].

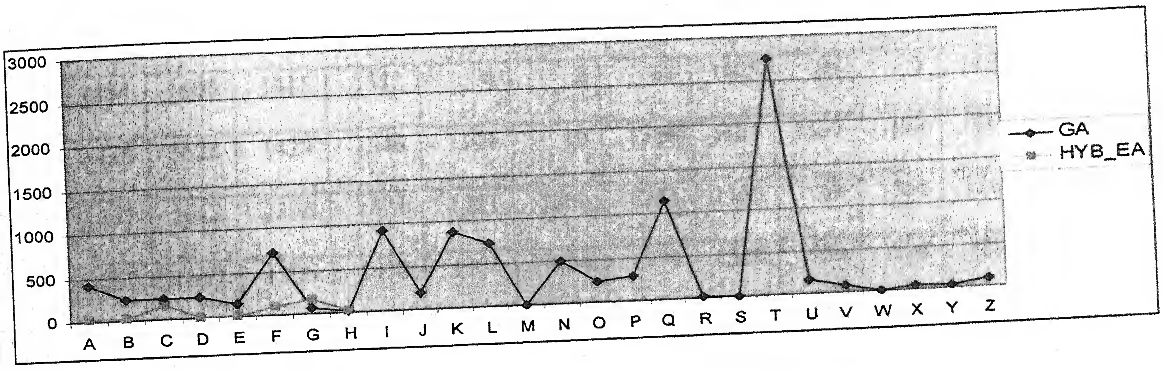


Table 4.10(a): Results third trial of for first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1235	1345	1334	1480	1302	1419	1457	1422	1651	1427	10	1114	1
Sample 2	58	1281	84	1314	1384	147	1342	1129	1534	1505	1547	1561	1327
Sample 3	1027	1482	1262	1295	1488	65	1662	1042	1688	1394	1553	1489	1278
Sample 4	990	1284	1248	1341	1543	103	1631	1067	1532	1552	260	1412	1417
Sample 5	1075	1163	1258	1362	1554	1306	1590	1295	120	1466	1268	51	1308
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1528	1757	1	1	3	1	20	1	The algorithm could not converge to the solution after alphabet 'H' so no convergence weight matrix obtained for remaining alphabets.				
Sample 2	67	1666	1944	1736	7	1862	23	1900					
Sample 3	1810	16	1940	1729	6	1861	1643	1897					
Sample 4	1838	1952	1955	1748	1693	1918	1707	1943					
Sample 5	1848	1943	1953	1737	1679	1866	36	1955					

Table 4.10(b): Results of third trial of for first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1434	1606	1415	1332	4	2	62	1405	95	5	1191	1291	9
Sample 2	1426	1628	1429	1346	194	10	1290	1405	1357	153	1260	1254	1291
Sample 3	1532	1612	1376	1421	1221	1400	1261	1336	26	1364	1264	1338	283
Sample 4	1533	1647	1326	1356	1300	1404	1379	1508	1539	1104	1218	1403	347
Sample 5	1530	1650	1427	1185	1245	1436	1533	1393	1574	1407	1265	1374	1361
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	The algorithm could not converge to the solution after alphabet 'H' so no convergence weight matrix obtained for remaining alphabets.												
Sample 2													
Sample 3													
Sample 4													
Sample 5													

Figure 4.12: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for third trial of simulation for network [4-5-5-5].

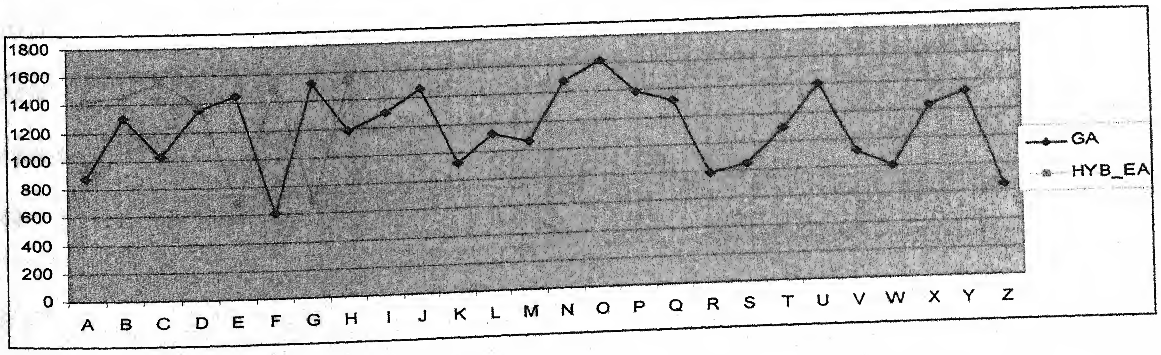


Table 4.11(a): Results of fourth trial of first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	3816	0.2	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	2159	492	286	1451	4983	169	80	5	4301	3146	6	8	747
Sample 2	116	1	2	2	1	1	1	13	2	2	2	1	1
Sample 3	1	1	649	1	1	1	1	1	2	2	2	1	1
Sample 4	2	1	1	1	6	1	54	1	11	2	18	1	210
Sample 5	61	1	1	1	1	1	2	1	1343	10	2	1	4
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	46	31	163	31	25	695	10	313	1	0.3	0.3	0.2	0.2
Sample 2	1	1	1	1	1	1	1	1	1	0.3	0.3	0.2	0.2
Sample 3	1	1	1	1	1	1	1	1	1	0.2	0.3	0.2	0.2
Sample 4	1	1	1	1	1	1	1	1	1	0.2	0.3	0.2	0.2
Sample 5	1	1	1	1	1	1	1	1	1	0.2	0.3	0.2	0.2

Table 4.11(b): Results of fourth trial of first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	158	672	40	2808	2	2	3	1	354	2	18	281	2
Sample 2	1	17	1	4	69	1	1	1	2	39	1	1	1
Sample 3	1	1	1	2	2	1	1	1	1	2	1	1	1
Sample 4	1	1	59	2	18	1	1	1	649	2	1	1	1
Sample 5	3	8	1	2	2	1	1	1	1768	2	1	1	1
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.3	0.1	0.2	0.3	0.3	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2

Figure 4.11: The Comparison Chart for iterations for recognition of handwritten English alphabets for fourth trial of simulation for network [4-5-5-5].

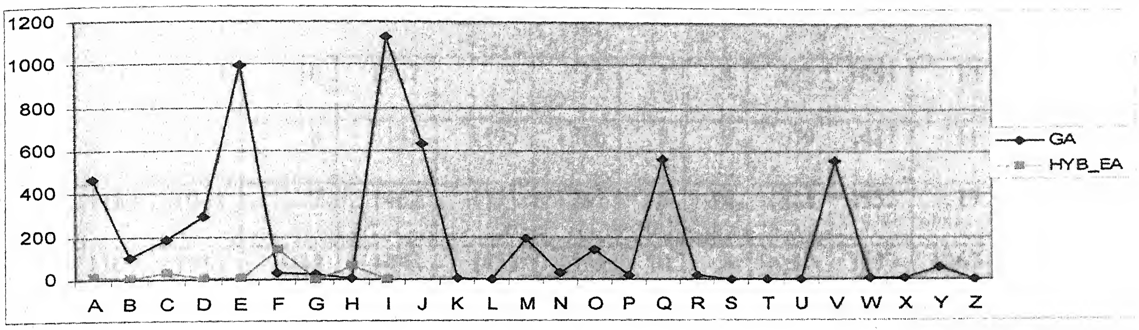


Table 4.12(a): Results of fourth trial of for first network [4-5-5-5].

lphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1277	1206	1476	1361	1480	1484	1371	9	1542	1638	1660	10	1299
Sample 2	1130	1378	1509	1393	1520	1359	199	9	1614	1574	1580	12	1483
Sample 3	1134	1254	124	1441	1570	1501	1284	14	1586	1516	1586	69	1241
Sample 4	1168	1345	25	1401	1590	1353	1668	10	103	1552	1729	160	1407
Sample 5	1136	1230	117	1489	1527	1495	1594	8	1484	1614	430	14	1486
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	119	27	1942	1944	1909	97	1748	1745	1	The algorithm could not converge to the solution after alphabet 'J' so no convergence weight matrix obtained for remaining alphabets.			
Sample 2	1667	1762	1922	1934	1902	1726	1744	1765	1664				
Sample 3	1632	1709	1941	1928	1890	1764	1750	1769	1689				
Sample 4	1685	1724	1942	1929	1888	1840	1741	1767	1666				
Sample 5	1627	1708	1922	1929	1875	1824	1755	1774	1664				

Table 4.12(b): Results of fourth trial of for first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1146	1494	10	1552	3	13	1	5	151	1493	10	1453	1
Sample 2	1100	1032	8	145	1195	1566	8	9	79	1427	11	1471	40
Sample 3	1184	1005	7	1462	1171	1281	6	10	1424	1552	17	1433	1515
Sample 4	1107	1013	1034	1443	1453	1596	10	8	10	1547	1267	1504	1359
Sample 5	1302	1093	107	1424	1293	1216	9	11	1481	1508	1331	1461	94
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	The algorithm could not converge to the solution after alphabet 'I' so no convergence weight matrix obtained for remaining alphabets.												
Sample 2													
Sample 3													
Sample 4													
Sample 5													

Figure 4.12: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for fourth trial of simulation for network [4-5-5-5].

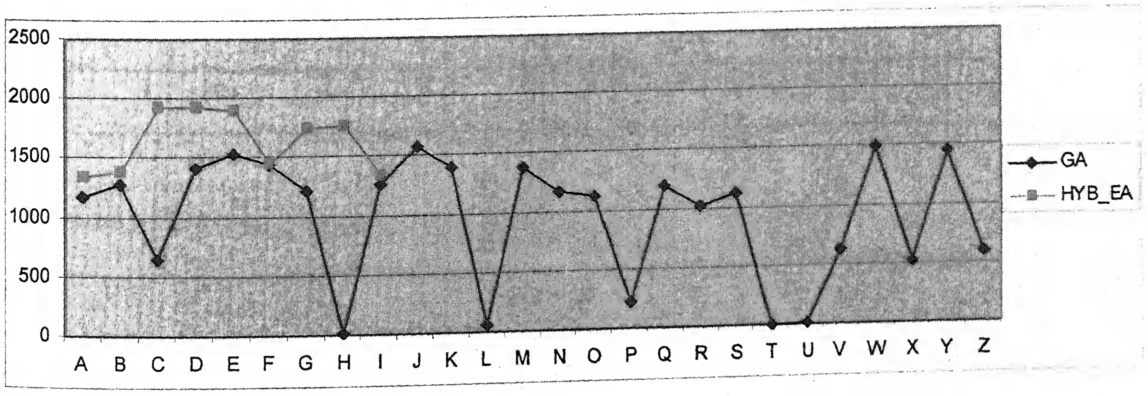


Table 4.13(a): Results of fifth trial of first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	183	0.2	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	1	0.4	0.3	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	1096	2114	1953	2539	9194	72	1428	1128	2	5	28	99	2
Sample 2	6	1	233	1	2	2	125	2	7	1	41	1	7
Sample 3	57	1	82	67	3	9	28	2	2	1	1	1	1
Sample 4	78	1	2	1	2	4	2	1	1	33	1	1	1
Sample 5	2	1	2	1	7	13	1	1	1	1	1	1	1
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	63	42	471	87	509	14	320	6	5110	4	3	3	20
Sample 2	2	2	91	2	2	1	2	1	1	2	2	1	1
Sample 3	2	2	1	2	14	1	2	1	1	2	2	225	1
Sample 4	2	2	2	2	2	1	2	1	1	2	1	1	1
Sample 5	2	261	2	2	2	1	2	1	1	2	2	1	1

Table 4.13(b): Results of fifth trial of first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	1450	91	1497	571	5137	231	578	2435	129	247	264	169	3914
Sample 2	1523	1	2	2	1	2	2	1	1	133	2	4	2
Sample 3	1	165	1	2	1	2	2	1	1	2057	131	1	7
Sample 4	1	1146	1	1	1	2	1	1	585	2	1	160	3
Sample 5	1	2	3	1	1	132	1	1	1	2	14	2883	5639
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	10	1	92	0.3	0.3	0.2	0.1	0.3	0.1	0.2	0.3	0.3	0.2
Sample 2	2	2	2	0.3	0.3	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.2
Sample 3	2	2		0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 4	2	39		0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2
Sample 5	2	1		0.3	0.3	0.2	0.2	0.2	0.2	0.3	0.2	0.3	0.2

Figure 4.13: The Comparison Chart for iterations for recognition of handwritten English alphabets for fifth trial of simulation for network [4-5-5-5].

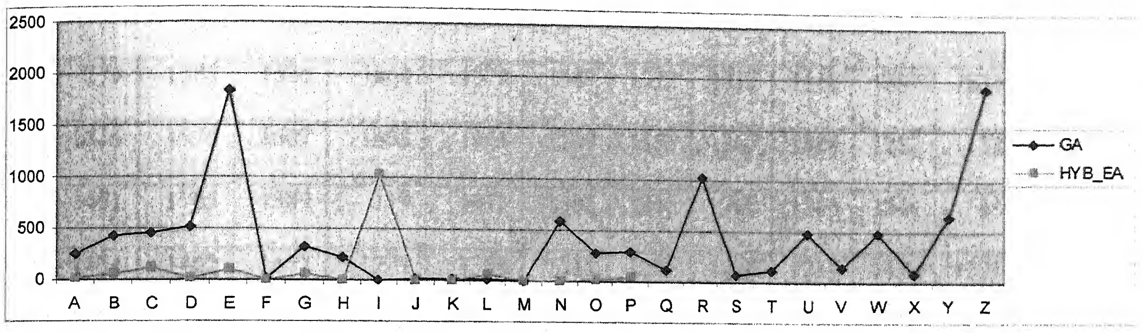


Table 4.14(a): Results fifth trial of for first network [4-5-5-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1155	1778	85	1212	1667	1667	1442	1410	4	6	1267	1358	11
Sample 2	91	1143	1579	1412	1652	1626	1492	1209	152	10	1291	1360	1316
Sample 3	71	63	1120	1505	1651	1623	1553	1181	1350	7	1172	1400	1342
Sample 4	184	167	63	1526	1594	1654	276	1251	45	1263	1373	1371	1379
Sample 5	916	94	1143	1385	1477	1518	82	1189	1392	1314	1205	94	1301
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1887	10	1646	1674	1674	1694	1713	1	1921	10	37		1498
Sample 2	1891	1831	1697	1698	1696	1698	1731	1864	111	1672	1671	1684	1584
Sample 3	1806	1853	1621	50	1613	1702	1705	1844	1753	73	1640	1529	39
Sample 4	1794	1866	1641	48	1610	1708	1728	1875	1709	84	2	2	1578
Sample 5	1876	1837	131	1641	1628	1674	1676	1856	141	1621	1659	1552	1553

Table 4.14(b): Results of fifth trial of for first network [4-5-5-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	273	1387	1526	1668	1372	1343	1273	1312	1512	1637	1031	1562	1497
Sample 2	1516	1575	1507	1653	1353	1199	1253	1363	1524	1287	138	1558	1521
Sample 3	1491	1705	1389	119	1412	1398	1246	1362	1544	1544	63	1575	1484
Sample 4	1534	1462	1477	1271	1352	1385	1207	1296	1231	1431	1271	38	1197
Sample 5	1494	1460	1374	1394	1380	91	1159	1322	1417	1430	1137	1324	71
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	121	1	1	The algorithm could not converge to the solution after alphabet 'P' so no convergence weight matrix obtained for remaining alphabets.									
Sample 2	1692	1696	1674										
Sample 3	1661	1684											
Sample 4	1686	1590											
Sample 5	1674	46											

Figure 4.14: The Comparison chart for convergence weight matrices for recognition of handwritten English alphabets for fifth trial of simulation for network [4-5-5-5].

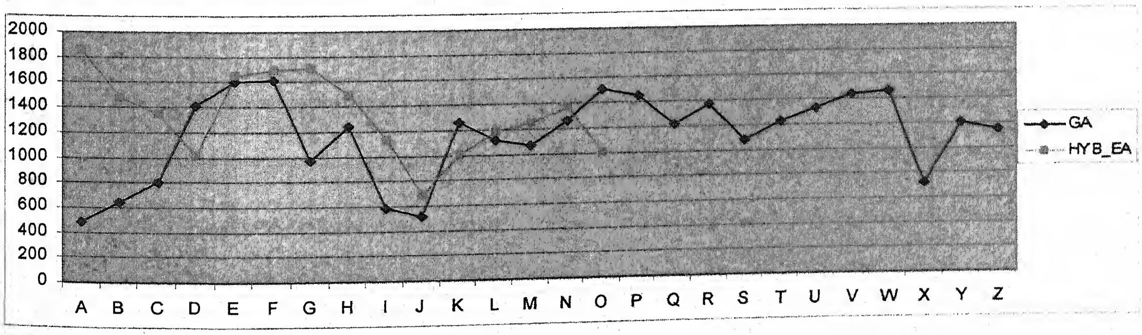


Table 4.15(a): Results of first trial of second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	4810	0.3	0.3	0.4	0.3	0.3	0.2	103	315	0.3	0.2	0.3	0.2
Sample 2	4	0.4	0.3	0.4	0.3	0.3	0.2	103	315	0.3	0.2	0.3	0.2
Sample 3	4	0.4	0.3	0.4	0.3	0.3	0.2	103	315	0.3	0.2	0.3	0.2
Sample 4	4	0.4	0.3	0.4	0.3	0.3	0.2	103	315	0.3	0.2	0.3	0.2
Sample 5	4	0.4	0.3	0.4	0.3	0.3	0.2	103	315	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	1556	15	52	295	25	11	17	123	37	46	8	96	12
Sample 2	22758	8	5	71	294	40	40	36	7	77	40	56	15
Sample 3	10	17	13	11	83	42	19	15	206	10	22	400	83
Sample 4	30	55	932	23	20	20	22	51	1	28	err	27	75
Sample 5	5203	706	241	31	4363	5	39	95	241	42	err	50	err
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	24	2	32	3	52	89	2	164	23	1	2	4	1
Sample 2	366	38	2	2	2	2	2	1	2	1	146	2	2
Sample 3	1	1	2	2	2	2	2	1	2	1	2	2	15
Sample 4	1	1	2	2	2	2	2	1	2	1	76	2	1
Sample 5	1	1	2	2	2	2	2	1	1	1	2	2	1

Table 4.15(b): Results of first trial of second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	0.1	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	51	22	43	22	159	37	6562	err	239	18	168	err	err
Sample 2	191	13	99	15	53	34	70	340	err	17	err	err	144
Sample 3	7	30	44	54	48	29	184	205	12	46	15	err	29
Sample 4	11	7	483	580	17	122	err	147	err	27	err	err	888
Sample 5	17	5	26	205	256	21	err	162	err	30	17	68	err
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	2	1	12	2	6	20	82	52	430	2	5	1	4
Sample 2	1	1	1	2	1	2	2	1	2	2	1	1	1
Sample 3	1	1	2	2	1	2	26	1	2	2	1	1	1
Sample 4	1	1	2	1	1	2	2	1	2	1	1	1	1
Sample 5	1	1	2	1	1	2	2	1	2	1	1	1	1

Figure 4.15: The Comparison Chart for iterations for recognition of handwritten English alphabets for first trial of simulation for network [4-6-6-5].

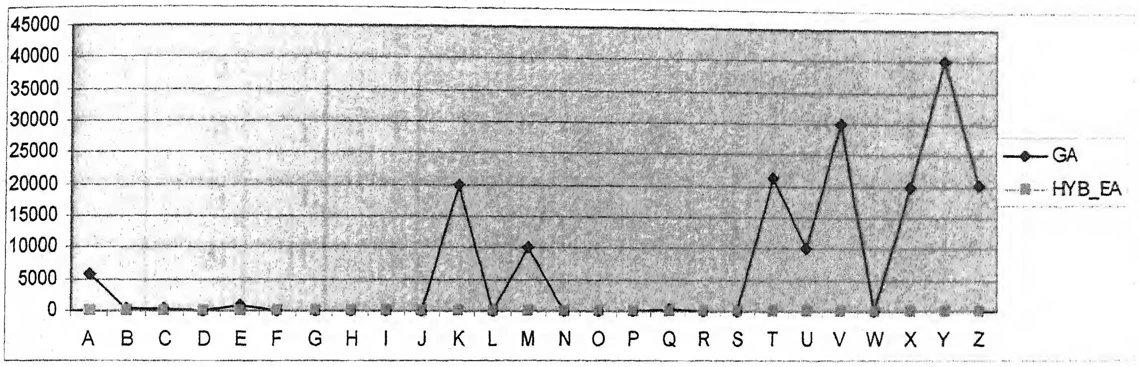


Table 4.16(a): Results of first trial of for second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1478	1	5	12	1	1	1	4	1	1	9	0	12
Sample 2	1		1	8	1	5	1	6	1	16	3	9	2
Sample 3	1	4	1	1	2	7	1	1	1	9	20	3	5
Sample 4	6		2	3	1	1	3	1	1	3		14	5
Sample 5	1		1	2	1	1	1	25	1	7		2	
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1762	1788	119	1	1919	1868	40	1785	1725	0	12	13	26
Sample 2	1836	43	1715	1943	1899	1827	1858	1846	1782	22	1831	1559	1706
Sample 3	1787	1947	1719	1934	1906	1838	1877	1805	1788	1683	1838	1584	1674
Sample 4	1754	1939	1722	1938	1917	1843	1831	1833	1774	108	1676	1585	1705
Sample 5	1792	1964	1709	1919	1906	1850	1856	1794	1770	1797	1696	1572	1690

Table 4.16(b): Results first trial of for second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1	2	1	1	1	0	3	0.5	1	78	1		
Sample 2	1	10	1	1	7	4	63	1		3			1
Sample 3	4	1	1	1	1	1	1	1	7	1809	34		15
Sample 4	6	24	11	3	1	1		1		5			1
Sample 5	5	1	5	2	1	1558		2		39	1	1	
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	10	21	1818	95	5	1835	1746	1794	1688	1680	1	10	37
Sample 2	91	1675	1783	1776	80	1851	60	1681	1692	1	301	1631	1617
Sample 3	1662	4	1814	1796	1725	1865	1754	1824	1683	1713	11	1573	1566
Sample 4	1607	1714	1785	108	1693	1816	1824	1679	1705	63	1626	23	1613
Sample 5	1677	1770	1743	134	1729	1829	1819	1806	1684	1690	1624	1567	1554

Figure 4.16: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for first trial of simulation for network [4-6-6-5].

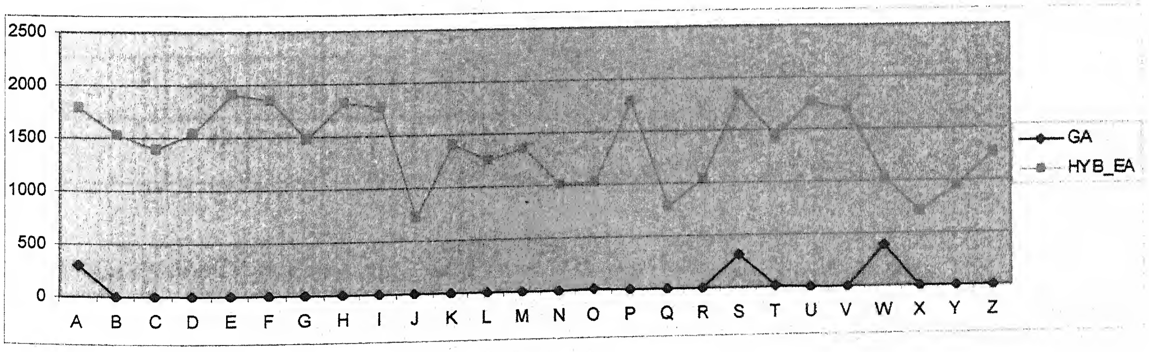


Table 4.17(a): Results of second trial of second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	25364	0.4	0.3	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	0.4	0.4	0.3	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	0.4	0.4	0.3	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	0.4	0.4	0.3	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	0.4	0.4	0.3	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	1707	301	4	3604	7254	2	7	34116	2100	33335	122	986	1
Sample 2	25	1	1	2	139	2	1	44	11	2	1	2	1
Sample 3	2	1	1	2	2	1	4	1	3	2	1	72	1
Sample 4	2	2	1	4335	253	1771	1	1	1	2	1	4	2
Sample 5	2	16	1	2	1	1	1	34	1	2	1	699	6
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	94	21	1	29	1	2	1	6	6	32	1274	99	2
Sample 2	2	2	1	3	50	1	2	1	1	1	2	1	1
Sample 3	2	2	1	1	2	1	2	1	1	1	2	1	1
Sample 4	2	2	1	1	2	1	1	1	1	1	467	38	1
Sample 5	1	32	1	1	2	1	1	1	1	1	83	2	1

Table 4.17(b): Results of second trial of second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	315	0.3	0.3	0.2	0.3	0.2	0.2	0.1	3222	0.2	2684
Sample 2	0.2	0.1	315	0.3	0.3	0.2	0.3	0.2	0.2	0.1	3222	0.2	2684
Sample 3	0.2	0.1	315	0.3	0.3	0.2	0.3	0.2	0.2	0.1	3222	0.2	2684
Sample 4	0.2	0.1	315	0.3	0.3	0.2	0.3	0.2	0.2	0.1	3222	0.2	2684
Sample 5	0.2	0.1	315	0.3	0.3	0.2	0.3	0.2	0.2	0.1	3222	0.2	2684
Iterations /Error calculated for Genetic Algorithm													
Sample 1	143	28	14	1483	304	647	500	3	1075	2	err	3744	16839
Sample 2	2	1	2	4	1	2	1	1	2	3	22323	4	2
Sample 3	2	1	1	1	1	1	1	1	2	2	2	1	2057
Sample 4	2	34	1	1	1	1	1	1	2	2	3331	341	1
Sample 5	2	52	1	2	1	1	1	1	9	1	23295	2186	1
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	35	41	3	55	1	6	3	3	864	60	1	1	187
Sample 2	139	1	1	2	2	2	1	2	1	2	38	1	2
Sample 3	1	1	1	40	2	2	1	2	1	2	1	1	2
Sample 4	1	1	1	2	2	27	1	2	1	2	1	1	3
Sample 5	1	1	1	2	2	2	1	2	1	2	1	1	1

Figure 4.17: The Comparison Chart for iterations for recognition of handwritten English alphabets for second trial of simulation for network [4-6-6-5].

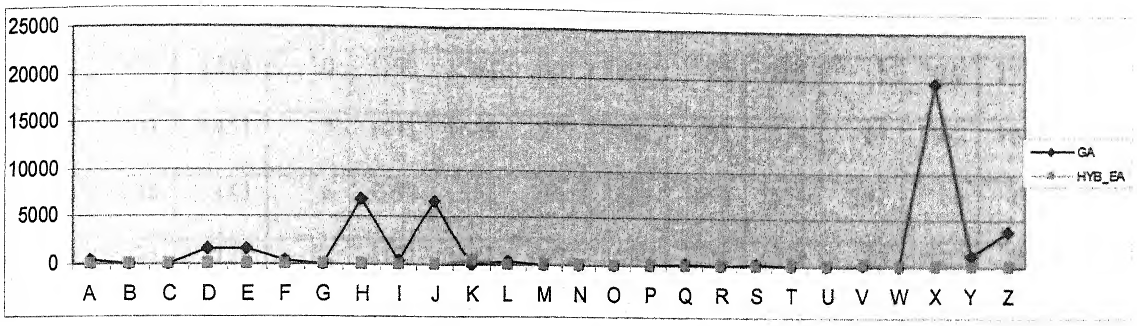


Table 4.18(a): Results of second trial of for second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	100	1384	9	1515	1730	1	189	1584	1718	24	1574	151	1
Sample 2	1435	1417	16	1555	1623	4	1718	1675	148	1702	1483	1542	3
Sample 3	1449	1346	1302	1514	1580	2	1754	1705	298	1744	1558	1564	5
Sample 4	1510	1396	1124	1649	1743	1415	1695	1627	1731	1783	1555	1508	1685
Sample 5		1278	1300	1651	1693	1371	1827	160	1505	1732	1565	1511	42
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1663	221	2	1732	43	2	22	2	1732	1809	1642	79	25
Sample 2	1795	199	1759	1889	85	1889	1722	1932	1632	1858	1618	1844	21
Sample 3	1775	1791	1770	1750	1774	165	1712	1886	1718	1818	1610	1801	1664
Sample 4	1825	1777	1720	1895	1791	1889	1825	1946	1672	1854	1805	64	1718
Sample 5	123	1632	1786	1849	1789	1887	1703	1872	1680	1846	1786	1765	1704

Table 4.18(b): Results of second trial of for second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	90	1294	10	1390	1382	1561	1595	22	1808	1	0.2	1712	325
Sample 2	1303	1451	7	1411	1649	1671	1592	291	1384	69	1362	1652	1504
Sample 3	148	143	6	1293	1553	1684	1624	1598	1374	107	66	180	1712
Sample 4	1324	124	12	1171	1621	1697	1529	1536	1478	37	1522	297	1793
Sample 5	1393	1656	5	94	1558	1699	1683	1603	1525	179	1699	1502	1778
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1761	1686	2	1892	0	1835	1	1	1787	40	0	110	1841
Sample 2	1807	1690	1833	1900	1	1861	145	1790	1786	1752	1772	53	1843
Sample 3	97	5	1849	1826	1731	1853	1805	1781	1784	1768	1789	1901	1825
Sample 4	1837	1659	1823	1846	1788	1879	1807	1804	1772	1739	1780	1801	1805
Sample 5	1737	1721	1821	1822	27	1863	1844	1781	1835	1756	1812	1892	1807

Figure 4.18: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for second trial of simulation for network [4-6-6-5].

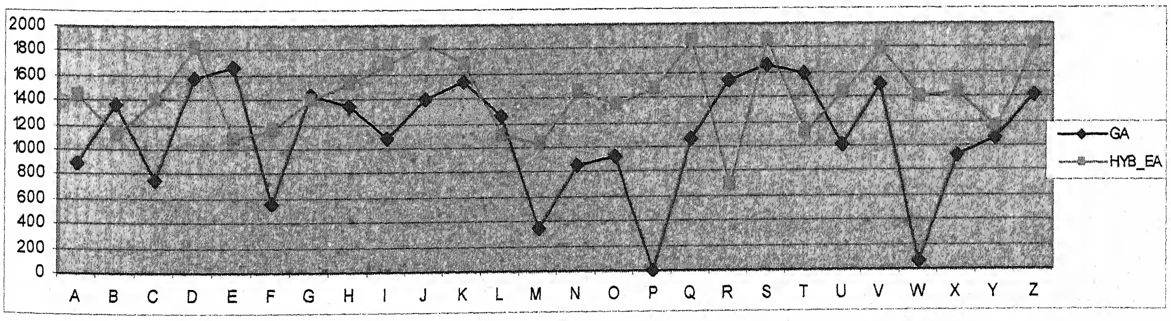


Table 4.19(a): Results of third trial of second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	32	721	482	0.3	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	3	15	432	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	3	15	1419	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	3	15	31	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	3	15	31	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	1584	9339	186	6	1821	1054	67	27	106	448	2	4632	1854
Sample 2	1849	404	18	24	2	2	2	1	2	2	2	1134	2
Sample 3	1	20	2	1	3441	6	38	1	313	2	2	2	2
Sample 4	1	47	6	1	4	680	2	1	10	2	229	2	2
Sample 5	1	2	4	1	2	10	2	1	6	2	108	1071	819
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	61	3	38	473	229	3	257	4	12	32	621	3	1
Sample 2	2	1	2	1	2	2	4	28	1	2	2	1	1
Sample 3	31	1	2	1	2	1	1	2	1	2	2	1	216
Sample 4	2	1	14	1	2	1	1	2	1	36	2	1	2
Sample 5	2	1	1	1	2	1	1	2	1	1	16	1	34

Table 4.19(b): Results of third trial of second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	0.4	0.3	0.3	315	0.3	0.2	0.2	3222	0.3	0.2	
Sample 2	0.2	0.1	0.4	0.3	0.3	315	0.3	0.2	0.2	3222	0.3	0.2	
Sample 3	0.2	0.1	0.4	0.3	0.3	315	0.3	0.2	0.2	3222	0.3	0.2	
Sample 4	0.2	0.1	0.4	0.3	0.3	315	0.3	0.2	0.2	3222	0.3	0.2	
Sample 5	0.2	0.1	0.4	0.3	0.3	315	0.3	0.2	0.2	3222	0.3	0.2	
Iterations /Error calculated for Genetic Algorithm													
Sample 1	1	92	163	711	2310	147	37520	565	459	68	6238	1094	5
Sample 2	2	2	6	2	2	1	1	126	2	2	10	1	
Sample 3	72	2	279	2	6	1	1	2	1	2	2	1	
Sample 4	2	2	11	112	4	1	1	2	1	289	102	1	
Sample 5	10	2	1	708	802	10	1	1	1	2	2	2	
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	198	6	40	5	22	1	203	1	26	93	3	1	
Sample 2	2	1	2	1	1	1	1	1	1	207	1	2	
Sample 3	42	1	39	1	1	1	1	1	1	2	1	2	
Sample 4	1	1	5	1	1	2	1	1	1	2	1	6	
Sample 5	1	1	1	1	1	1	1	1	1	2	1	1	

Figure 4.19: The Comparison chart for iterations for recognition of handwritten English alphabets for third trial of simulation for network [4-6-6-5].

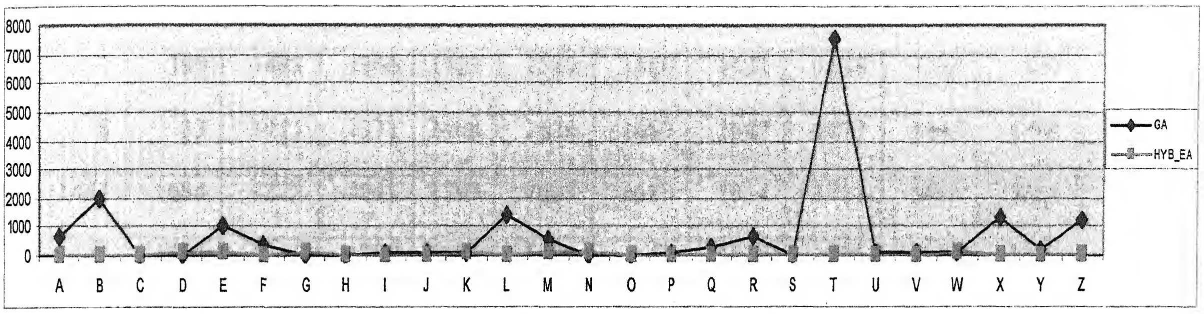


Table 4.20(a): Results of third trial of for second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1205	1602	1348	11	1449	11	1728	4	153	1639	5	1759	14
Sample 2	1463	1565	61	1226	1547	1555	1727	42	1277	1623	1709	1737	15
Sample 3	206	1509	1499	1170	1434	1725	28	206	113	1571	1684	1711	14
Sample 4	1466	1609	1573	1371	1354	1581	1687	1270	1434	1652	1549	1745	14
Sample 5	23	1484	1567	1145	1377	1721	1690	1420	1343	1637	1695	1649	17
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	62	2	136	1923	69	13	50	1788	1810	1819	1786	2	1
Sample 2	1508	1717	1832	1930	1755	1808	1750	1816	1784	1727	1857	1723	1
Sample 3	1661	1719	1800	1919	1774	4	1709	1806	1842	1764	1878	1790	
Sample 4	1669	1690	1639	1928	1744	1830	1802	1768	1802	1755	1869	1730	18
Sample 5	1780	1709	1711	1924	1766	1850	1762	1713	1844	1771	1915	1797	18

Table 4.20(b): Results of third trial of for second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1	199	1488	1366	1523	1519	1580	1521	1638	1539	155	1623	1706
Sample 2	8	33	1411	1371	1408	1626	1687	1647	1777	1642	1596	1588	1527
Sample 3	1451	1654	1222	1343	1397	1608	1619	1614	1721	1623	1661	1653	1516
Sample 4	1441	1591	1219	1298	1485	1557	47	1516	1786	88	1589	1553	1552
Sample 5	1609	1635	1213	1623	1521	1565	1488	1460	1719	1456	1626	1602	1457
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	87	2	1869	328	126	86	1874	29	1833	1801	21	57	1780
Sample 2	1787	1853	1857	1852	1834	1876	44	58	1877	1786	41	1675	1651
Sample 3	1782	1812	1822	1889	1840	1882	64	1739	1817	1754	1746	1812	1763
Sample 4	47	1824	1809	1910	1830	1904	85	1712	1844	1754	1741	1878	1705
Sample 5	1711	1829	1824	1904	1826	1900	1778	1718	1821	1749	1847	1903	1753

Figure 4.20: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for third trial of simulation for network [4-6-6-5].

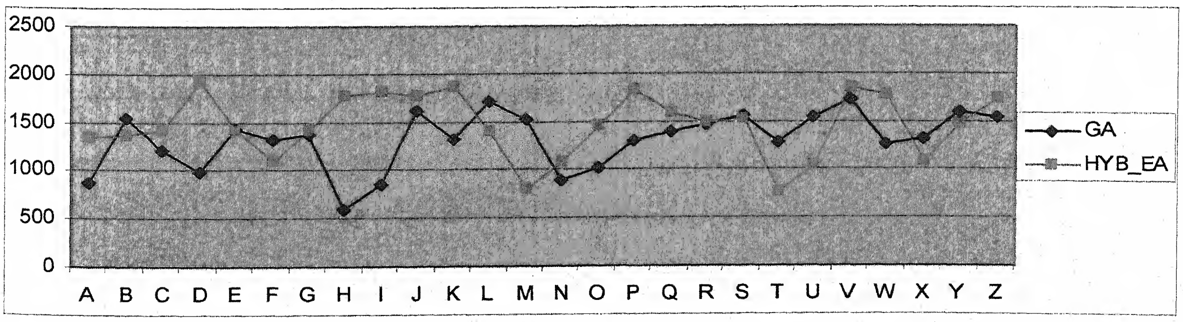


Table 4.21(a): Results of fourth trial of second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	32	75	2629	5953	0.3	103	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	3	3	17	3	0.3	103	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	3	3	17	3	0.3	103	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	3	3	17	3	0.3	103	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	3	3	17	3	0.3	103	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	2038	175	1471	535	11256	1965	868	282	4264	3	175	7	89
Sample 2	20	2	148	2	2	1	4	2	4	3	2	135	17
Sample 3	2	7	9	1011	2	4	1	198	2	4	2	3	2
Sample 4	2	2	1	1	2	3176	1	2	2	1	359	461	2
Sample 5	2	15	2	1	2	21	1	2	74	2	4	2	54
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	35	33	846	51	173	42	2	56	70	2	2	462	1
Sample 2	2	2	1	2	2	2	2	1	3	2	1	2	2
Sample 3	2	2	1	2	2	192	2	1	2	1	1	2	2
Sample 4	4	2	1	198	2	1	2	1	2	1	2	2	2
Sample 5	2	2	1	2	2	2	2	1	2	1	8	2	2

Table 4.21(b): Results of fourth trial of second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	315	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 2	0.2	315	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 3	0.2	315	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 4	0.2	315	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Sample 5	0.2	315	0.4	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.3	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	606	42	25	2	5886	1197	19588	6099	23641	90	247	490	5001
Sample 2	1	1	1	3	1	75	875	2	2	2	1	1	2
Sample 3	28	1	1	96	1	2	482	2	7	2	1	1	462
Sample 4	2	1	1	28	1	359	6	3669	2	2	1	1678	2
Sample 5	2	1	1	2	1	8	2	12	1	2	1	1	2
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	511	73	24	1	1	1	3	1	38	1	13	121	57
Sample 2	760	2	40	1	1	1	2	1	1	1	2	1	1
Sample 3	2	2	1	1	1	1	2	1	1	2	2	1	1
Sample 4	1	3	1	1	1	1	2	1	1	2	2	1	1
Sample 5	1	2	1	1	1	1	2	1	1	1	2	1	1

Figure 4.21: The comparison chart for iterations for recognition of handwritten English alphabets for fourth trial of simulation for network [4-6-6-5].

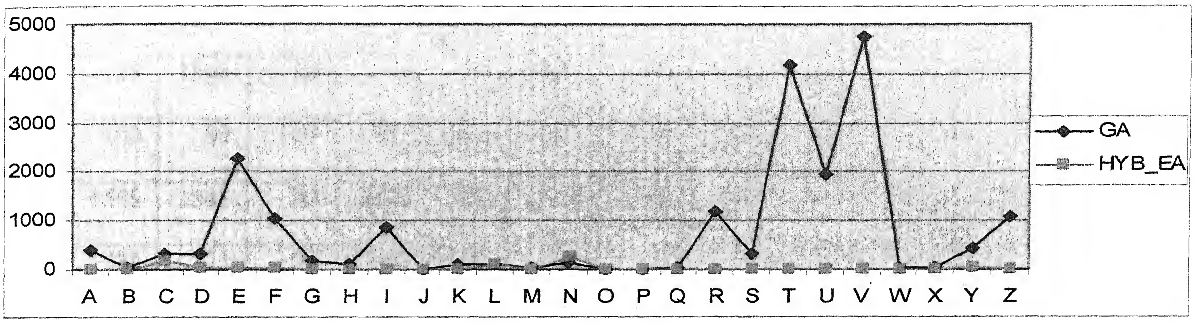


Table 4.22(a): Results of fourth trial of for second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1409	1387	1318	231	1796	1492	1230	36	1624	1	1676	9	1172
Sample 2	113	1417	1348	112	1780	1267	1338	68	1580	11	1723	340	121
Sample 3	1185	1395	1318	1282	1771	1433	1323	1403	1685	1602	1716	1491	1558
Sample 4	1176	1392	51	1236	1767	1468	30	1395	1591	40	1229	1454	1621
Sample 5	1216	140	1313	1397	1721	1574	1391	1431	1513	1473	1598	1389	17
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1	1	1789	1868	1907	1787	72	1891	1781	8	1	1901	30
Sample 2	5	1844	1799	1859	1916	1889	1829	1899	1808	1811	1872	1835	1763
Sample 3	1	1785	1806	1870	1918	1843	1827	899	1826	1804	1802	1774	1682
Sample 4	14	1760	1816	1919	1899	1838	1839	1899	1803	1869	1815	1757	1747
Sample 5	13	1784	1722	1807	1931	1849	1847	1923	1835	1800	1808	1787	1691

Table 4.22(b): Results of fourth trial of for second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	1129	1566	12	3	11	1666	53	108	1559	20	1388	1629	1678
Sample 2	100	80	24	1521	1530	1631	1660	183	1707	1774	1519	1557	1676
Sample 3	1509	1543	45	1385	1469	1498	128	212	154	1775	1580	1709	136
Sample 4	1284	225	21	1564	1561	140	1512	1735	1298	1783	1465	234	1485
Sample 5	1370	1554	1333	1559	67	1759	1517	1602	1699	1802	1538	152	1506
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1766	1732	1937	4	3	64	6	116	1822	72	1766	1801	1804
Sample 2	1777	1634	1888	113	1815	41	1605	104	1792	79	1762	1816	1854
Sample 3	1751	1649	1927	1759	1848	1817	1648	123	1811	1699	1767	1876	1770
Sample 4	1682	1831	1926	1777	1788	1743	1635	1698	1791	1685	1756	1895	1809
Sample 5	1729	1822	1937	1779	1829	1792	1638	1734	1820	1709	1782	1857	1812

Figure 4.22: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for fourth trial of simulation for network [4-6-6-5].

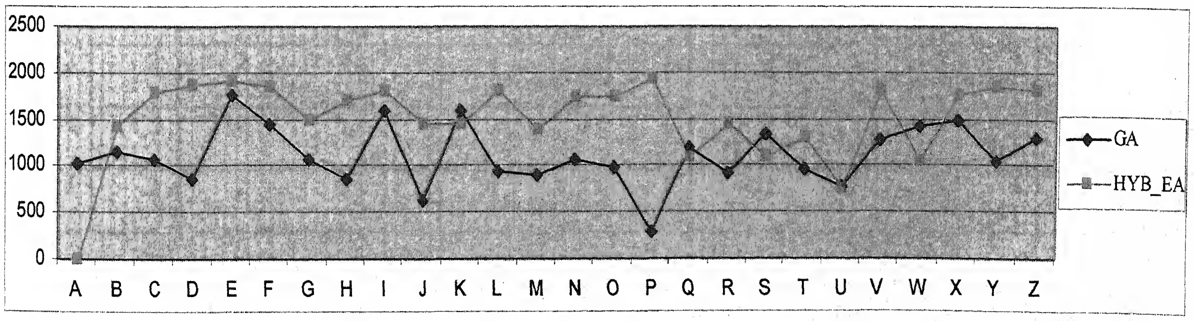


Table 4.23(a): Results of fifth trial of second network [4-6-6-5].

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Iterations / error calculated for Back-Propagation Algorithm													
Sample 1	94	1459	222	0.2	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 2	31	23	17	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 3	31	23	17	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 4	31	17	17	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Sample 5	155	17	17	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2	0.3	0.2
Iterations / error calculated for Genetic Algorithm													
Sample 1	1956	166	11940	30	54	5419	2	19941	2203	101	223	358	412
Sample 2	24	3717	1886	2	1	1	2	7070	2	1	2	7	688
Sample 3	2	11	15	2	1	1	91	4	2	1	8	1	2
Sample 4	15	208	2	2	2	1	437	2	2	1	2	1	140
Sample 5	2	2	2	2	3	1	4	4	2	1	2	1	1
Iterations / error calculated Hybrid Evolutionary Algorithm													
Sample 1	82	97	1207	2	1	1	1	53	3	9	378	2	279
Sample 2	2	2	1	1	1	1	1	2	1	1	1	1	1
Sample 3	2	2	1	1	1	1	1	2	1	2	1	1	1
Sample 4	2	210	1	1	1	1	1	2	1	2	1	1	1
Sample 5	2	2	1	1	1	1	1	2	604	2	1	1	1

Table 4.23(b): Results of fifth trial of second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Iterations /Error calculated for Back-Propagation Algorithm													
Sample 1	0.2	0.1	252	0.3	0.3	0.2	0.3	0.2	0.2	0.1	103	0.2	0.2
Sample 2	0.2	0.1	252	0.3	0.3	0.2	0.3	0.2	0.2	0.1	103	0.2	0.2
Sample 3	0.2	0.1	252	0.3	0.3	0.2	0.3	0.2	0.2	0.1	103	0.2	0.2
Sample 4	0.2	0.1	252	0.3	0.3	0.2	0.3	0.2	0.2	0.1	103	0.2	0.2
Sample 5	0.2	0.1	252	0.3	0.3	0.2	0.3	0.2	0.2	0.1	103	0.2	0.2
Iterations /Error calculated for Genetic Algorithm													
Sample 1	159	135	440	9024	23	3	45	3942	724	1124	3831	8	1117
Sample 2	1	2	36	2	2	2	1	4	1	2	1	1	1
Sample 3	1	4	2	2	2	2	1	1	1	8	2	1	1
Sample 4	1	2	344	1	48	7	64	1	1	2	22682	1	1
Sample 5	1	2	8	8	2	2	2	1	1	2	662	2	1
Iterations /Error calculated for Hybrid Evolutionary Algorithm													
Sample 1	2	1	23	215	12	507	17	3	3	81	218	145	25
Sample 2	1	1	1	1	1	1	17	2	1	1	1	1	2
Sample 3	1	1	1	1	1	1	30	12	1	1	1	1	21
Sample 4	1	1	1	1	1	1	26	1	1	1	1	1	2
Sample 5	1	1	1	1	1	1	1	1	1	1	1	1	2

Figure 4.23: The comparison chart for iterations for recognition of handwritten English alphabets for fifth trial of simulation for network [4-6-6-5].

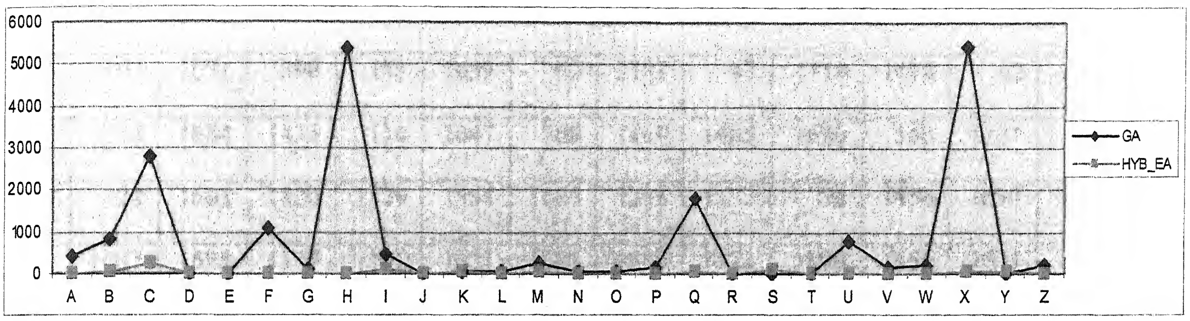


Table 4.24(a): Results of fifth trial of for second network [4-6-6-5].

ıabets	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of convergence weight matrix for Genetic Algorithm													
ample 1	1004	1445	1531	1540	1395	1537	4	3	1725	21	1523	1701	1489
ample 2	1165	37	1581	1382	1364	1499	207	1674	1709	1524	1589	1691	1452
ample 3	1037	1486	1560	1368	1321	67	45	1544	1729	1582	1479	62	1500
ample 4	1226	1282	1526	1525	1213	1452	1491	1536	1701	1516	1679	1684	1372
ample 5	1339	1354	1535	1614	158	38	1645	1462	1579	1615	1615	1443	1531
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
ample 1	1731	1964	1735	1	11	1	10	1864	1684	1808	1649	16	1679
ample 2	1760	1957	1698	1894	48	1919	1732	1928	1666	1945	1639	1732	1786
ample 3	1735	1966	1767	1911	1694	1911	1719	1852	1650	1943	1680	1819	1727
ample 4	1742	1923	1731	1855	1697	1917	1705	1809	1662	1935	28	1729	1718
ample 5	1763	1861	1688	1887	1680	1891	1760	1878	1741	1885	1753	1811	1698

Table 4.24(b): Results fifth trial of for second network [4-6-6-5].

Alphabets	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number of convergence weight matrix for Genetic Algorithm													
Sample 1	107	1671	398	141	1652	62	1382	67	1119	1618	62	1516	1091
Sample 2	112	1684	1435	114	1647	208	1469	1485	1630	168	1721	46	1373
Sample 3	128	1663	1520	1579	1598	1627	1285	1172	78	1556	1690	1591	1332
Sample 4	1467	1695	1377	1795	1631	1681	1666	1473	1699	1655	1504	1593	204
Sample 5	151	1722	26	1803	1624	1635	1485	1388	1275	1631	1678	1538	1328
Number of convergence weight matrix for Hybrid Evolutionary Algorithm													
Sample 1	1	30	1755	1689	1774	1659	90	160	1	1788	1757	1734	1835
Sample 2	1764	1690	1817	1783	1743	1670	1799	1711	116	1828	1783	43	1806
Sample 3	1757	1755	1750	1722	1773	1680	1710	1757	1835	1782	1827	52	1804
Sample 4	1758	1712	1712	1779	1697	1695	1825	1706	1768	1807	1815	1768	1843
Sample 5	1745	1742	1807	1791	1795	1719	1796	1782	1827	1761	1826	1686	1803

Figure 4.24: The Comparison Chart for convergence weight matrices for recognition of handwritten English alphabets for fifth trial of simulation for network [4-6-6-5].

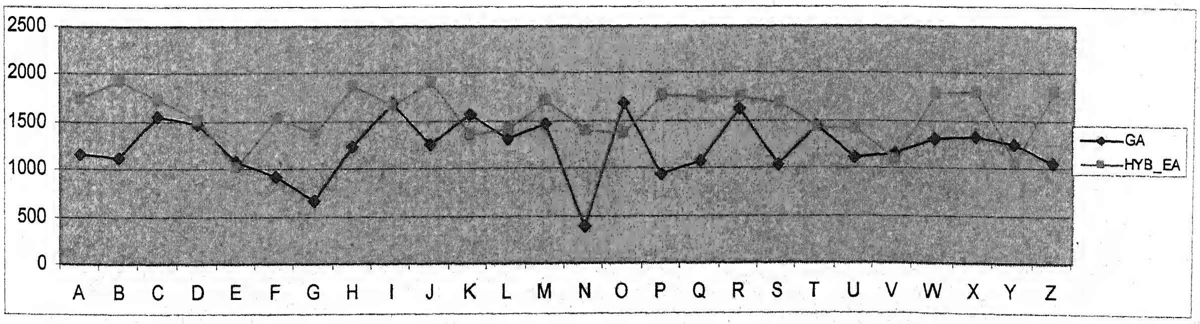


Table 4.25: Average iterations of all alphabets for each trial of second network [4-6-6-5].

Average Iterations	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
GA	6931	1688	687	767	782
Hyb_EA	14	30	24	31	34

Figure 4.25: Comparison chart for Average iterations for all alphabets for each trial of second network [4-6-6-5].

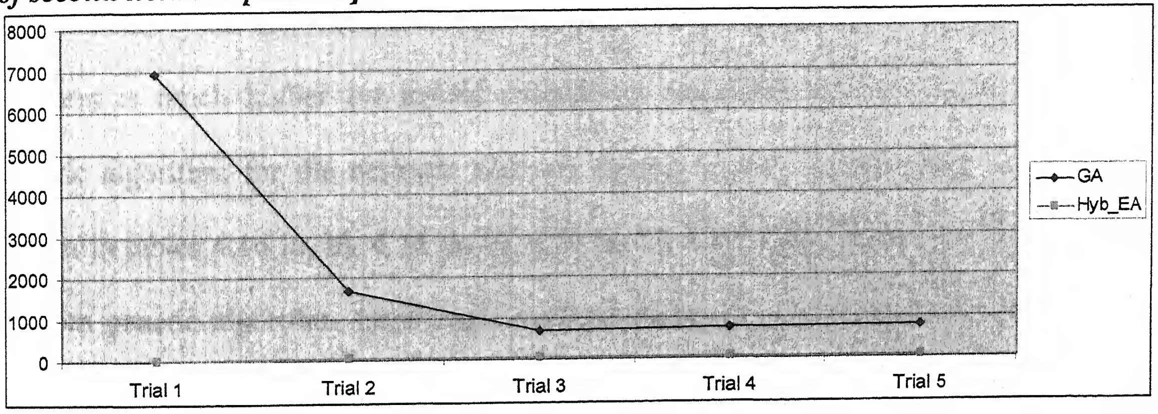
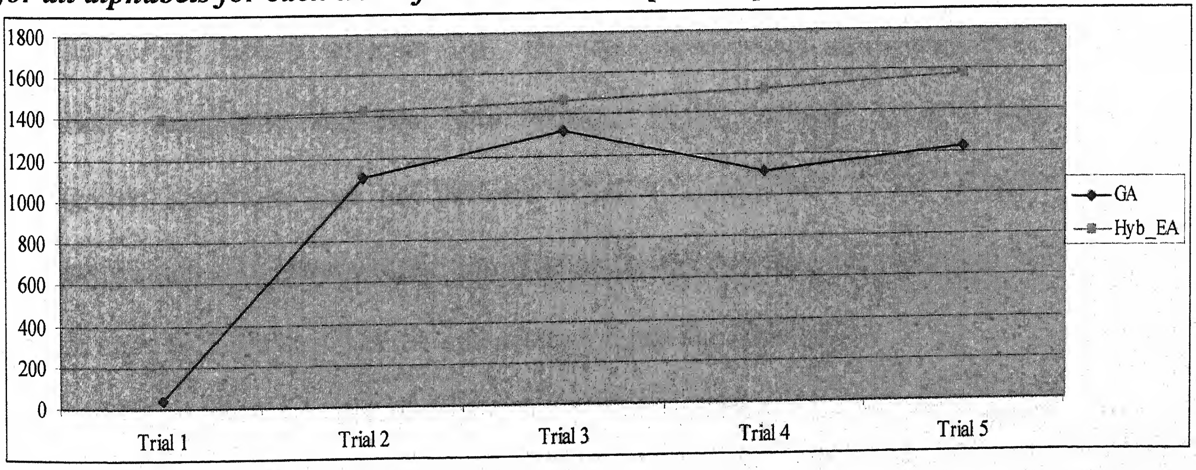


Table 4.26: Average number of convergence weight matrices of all alphabets for each trial of second network [4-6-6-5].

Average Counts	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
GA	42	1113	1323	1116	1232
Hyb_EA	1393	1430	1466	1513	1577

Figure 4.25: Comparison chart for Average number of convergence weight matrices for all alphabets for each trial of second network [4-6-6-5].



4.6 Conclusion

Results of the experiments clearly show that hybrid evolutionary algorithm takes less number of iterations in comparison to simple genetic algorithm, though it is found that the back-propagation algorithm does not conclude the results for this problem up to the 50000 iterations, however it suffers with the local minima of unknown error surface for this problem. It has also been found that the rate of convergence of weight matrix for this problem is much higher for hybrid evolutionar algorithm in comparison to random genetic algorithm for the network with six neuron in two hidden layers [4-6-6-5] as shown in tables 4.16 [a, b], 4.18 [a, b], 4.20 [a, b], 4.22[a, b], 4.24[a, b] . However the random genetic algorithm surprisingly perform better in case of trials of network with five neurons in two hidden layers[4-5-5-5] in comparison to hybrid evolutionary algorithm, which could not converge after the recognition of 4-5 alphabets as shown in figure 4.6 [a, b], 4.8 [a, b], 4.10 [a, b], 4.12 [a, b], 4.14 [a, b]. These values show that many numbers of convergence weight matrices have been obtained for the particular character by applying the genetic algorithm and hybrid genetic algorithm. So on the basis of the above mentioned results the following conclusion can be made:

1. The results clearly indicate that, for the classification of handwritten English alphabets recognition problem, feed-forward neural network trained with back-propagation algorithm does not perform better in comparison to feed-forward neural network trained with genetic algorithm. The performance of genetic algorithm and hybrid genetic algorithm are found better and accurate in all the simulations.
2. It is found that, for the network [4-6-6-5], the hybrid evolutionary feed-forward neural networks gives more than one convergent weight matrices for every input pattern

in comparison to the back-propagation feed-forward neural network and random genetic algorithm. This shows the higher accuracy rate in the pattern recognition with hybrid evolutionary feed-forward neural network.

3. The higher number of convergence weight matrices in the hybrid EA training process suggests that this algorithm may not be trapped in the false minima of the error surface. But it is also found surprisingly that the first network consisting of five neurons in two hidden layers is also being trapped with the local minima problem in case of hybrid evolutionary algorithm. It may also increase the possibilities of misclassification for any unknown testing input pattern.

4. The performance of hybrid evolutionary algorithm is found better in comparison to the genetic algorithm by finding the less number of iterations as shown in the table 4.25 & Figure 4.25 and higher rate of convergence for character recognition in comparison to the results for simple genetic algorithm as shown in the table 4.26 & Figure 4.26.

5. The network consist two hidden layers with five neurons is trapped in local minima of previously learned pattern's error surface in case of hybrid evolutionary algorithm. It is observed that after the recognition of 6-7 character the network is fail to converge for the remaining alphabet samples. It means the network is suffering from the tendency to occupy the local minima of error surface for the previously trained pattern. It is not in the case of second network consisting six neurons in two hidden layers [4-6-6-5]. The second network is converged for almost every sample of alphabets. So it may also be concluded that the network consisting six neurons in two hidden layers are provide the optimum network.

References

[1] Jain, A.K., “Statistical Pattern Recognition: A Review.”, IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 22(1), pp. 4-37,(2000).

[2] Casey, R.G., and Ferguson, D.R., “Intelligent Forms Processing.” ,ZBM Syst. J.,vol. 29, pp. 435-450, (1990).

[3] Christenson,P., Maurer, A., and Miner,G.,” Handwriting recognition by neural network.”,(2005). (http://csci.mrs.umn.edu/UMMCSciWiki/pub/CSci4555s04/InsertTeamNameHere/handwriting.pdf.).

[4] Friedman, M., and Kandel, A.,” Introduction to Pattern Recognition: Statistical,Structural, Neural, and Fuzzy Logic Approaches.”, World Scientific, Singapore, (1999).

[5] Duda, R.O., Hart P.E., and Stork, D.G.,” Pattern classification (2nd edition).”, Wiley, New York, ISBN 0-471-05669-3, (2001).

[6] Jensen, F.V. ,”Bayesian Networks and Decision Graphs.” Springer. (2001).

[7] Cover, T. and Hart, P. ,” Nearest Neighbor Pattern Classification.”, IEEE Trans. on Information Theory, vol. 13(1),pp. 21-27,(1967).

[8] Wu,M.P.H., ”Hand Written Character Recognition”, The school on Information Technology and Electrical Engineering, University of Queensland,(2003).

[9] Muller, B. and Reinhardt, J. ,”Neural Networks: An Introduction.”, Physics of Neural Network, New York: Springer-Verlag, (1991).

[10] Barto, A.G., Sutton, R.S. and Anderson, C., "Neuron-like adaptive elements that can solve difficult learning control problems,", IEEE Transaction on Systems, Man and Cybernetics, vol.13, pp. 834-846,(1983).

[11] Dayhoff, J.," *Neural-Network Architectures: An Introduction.*" New York: Van Nostrand Reinhold, (1990).

[12] Grossberg,S. "Some networks that can learn, remember and reproduce any number of complicated space – time patterns", J. Math. Mech., vol.1(19), pp. 53 – 91 (1991).

[13] Sutton, R.S., Barto, A.G., and Williams, R.J.," Reinforcement learning is direct adaptive optimal control.", IEEE Control Systems Magazine, vol. 12,pp. 19 – 22,(1999).

[14] Sontag, E.D., "Feed-forward nets for interpolation and classification", J. Computing System Sciences, vol. 45, pp. 20-48,(1992).

[15] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning internal representations by error propagation.", Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1,pp. 318 – 362, (1986).

[16] Kim, H.B., Jung, S.H., Kim, T.G. and Park, K.H., "Fast learning method for backpropagation neural network by evolutionary adaptation of learning rates.", Neurocomputing, vol. **11**(1), pp. 101–106, (1996).

[17] Yahya, H.Z., Lakmal, D. and Seneviratne,K.A.," Stability analysis of a three-term backpropagation algorithm" Neural Networks, vol. 18(10), pp. 1341 – 1347,(2005).

[18] Sprinkhuizen, I.G.K., Egbert J.W.B., "The local minima of the error surface of the 2-2-1 XOR network." Annals of Mathematics and Artificial Intelligence, vol 25(1-2), pp. 107 - 136, (1999).

[19] Mangal,M. and Singh,M.P., "Analysis of pattern classification for the multidimensional parity-bit-checking problem with hybrid evolutionary feed-forward neural network.", Neurocomputing, volume 70,pp. 1511-1524,(2007).

[20] Mangal M. and Singh,M.P., 'Analysis of multidimensional XOR classification problem with evolutionary feed-forward neural networks.", International Journal on Artificial Intelligence Tools , volume 16(1),pp. 111-120,(2007).

[21] Yegnarayana, B.,"Artificial Neural Networks", Prentice Hall of India, (2004).

[22] Du K.L. and Swamy, M.N.S.," Neural Networks in a Soft computing Framework ", Springer-Verlag London Limited ,(2006).

[23] Goldberg, D., "Genetic Algorithms in Search, Optimization, and Machine Learning Reading", MA: Addison-Wesley Publishing Company, Inc., (1989).

[24] H Holland, J., "Adaptation in natural and artificial systems." ,University of Michigan Press, Ann Arbor, Michigan, (1975).

CHAPTER 5

Analysis for the Recognition of Scaled and Rotated Hand Written English Alphabets with Online Learning Implementation using Soft-Computing Techniques

Chapter 5: Analysis for the Recognition of Scaled and Rotated Hand Written English Alphabets with Online Learning Implementation Using Soft-Computing Techniques.

Abstracts

This chapter describes the performance of online learning capabilities of neural networks with the genetic algorithm and hybrid evolutionary algorithms for hand written English alphabets after their scaling and rotation. The random genetic algorithm and hybrid evolutionary algorithm are used for training to recognize the scaled and rotated alphabets. Genetic algorithms for the hybrid neural network are showing the numerous potential in the field of pattern recognition. We have taken two samples of each hand written English alphabets, one trained sample and another one is fresh or unknown sample, after applying the general mathematical algorithm for rotation and scaling of the sample on both X and Y axis. The random genetic algorithm and the hybrid evolutionary algorithm are applied to train these samples. Network trained for straight alphabet samples have been used to recognize the rotated and scaled sample from the set of already trained five straight alphabet's sample. The online learning algorithm with the combination of evolutionary algorithm has been taken the definite lead on the conventional approaches of neural network and soft computing techniques. The results of the experiments clearly show that the hybrid approach is efficient to train the hand written samples of any shape and size most reliably.

5.1 Introduction

For more than thirty years, researchers have been working on handwriting recognition. This new stage in the evolution of handwriting processing results from a combination of several elements: improvements in recognition rates, the use of complex systems integrating several kinds of information, the choice of relevant application domains, and new technologies such as high quality & high speed scanners and inexpensive powerful processors[1]. Methods and recognition rates depend on the level of constraints on handwriting. The constraints are mainly characterized by the types of handwriting, the number of ***scripter,*** the size of the vocabulary and the spatial layout. Recognition strategies heavily depend on the nature of the character set to be recognized. Recognition becomes more difficult when the constraints decreases. Intense activity was devoted to the character recognition problem during the seventies and the eighties and pretty good results have been achieved [2]. Character recognition techniques can be classified according to two criteria: the way preprocessing is performed on the data and the type of the decision algorithm. There have been reports of successful use of neural networks for the recognition of handwritten characters [3-5], but it is very difficult to find any general investigation which might shed light on the systematic approach of a complete neural network system for the automatic recognition of cursive character.

Traditionally, the recognition of hand-written characters has been considered of importance for various applications in which the automatic reading of hand written text is needed. More recently, the recognition of hand-written characters has gained importance to develop the ***intelligent system.***

The hand written characters may have the different shapes and different size .It may be straight, scaled and / or rotated. The human can easily recognize the character of any shape and size. But for the machine every alphabet after scaling or/ and rotation becomes new pattern. So the recognition of the hand written character is very difficult-indeed, the symbols can be of any size and orientation in the image frame. The recognition rate for rotated and / or scaled hand written characters cannot be as good as the straight characters.

The recognition rate depends on the number of writers and their training. If the number of writers is more, recognition can be more difficult. There are several techniques to recognize the hand written character which are in the straight form but a very few work have been done for scaled and rotated hand written alphabets [6-10]. A feature based approach is also suggested that are not sensitive to rotations, translations, and scaling, and are resistant to noise and distortions as well. Some works have been done on Arabic cursive character [11, 12]. Nawwaf & Rabab [11] also proposed a simple and computationally efficient mapping, which can be used for character recognition by taking the application of hand-written Arabic characters, and explained that it (taken with other features of the character) can be used to produce an effective recognition system to identify each character uniquely. The recognition process for a large set of complex problems such as hand written characters after scaling and rotation mostly depends on the way of training. Efficient learning of a pattern largely depends upon the training methods. The process of learning of the pattern may be divided into two basic principals 'on-line learning' and 'off-line' learning. In an off-line learning the given patterns are used together to determine the weights. On the other hand in an on-line learning the information in each new pattern is incorporated into the network by incrementally adjusting the weights [13]. Thus, an on-line learning allows the network to update the information continuously. The online learning is one of the most powerful and commonly used techniques for training large layered networks and large training set, and has been used successfully in many real-world applications. The powerful combination of analytical methods provides more insight and deeper understanding of existing algorithms, and leads to novel and principled proposals for their improvement. The results of the handwritten English straight alphabets recognition problem clearly shows that,[14] feed-forward neural network trained with back propagation algorithm does not perform better in comparison to feed forward neural network trained with genetic algorithm . The performance of genetic algorithm and hybrid evolutionary algorithm are found better and accurate in all the simulations. The higher number of convergence weight matrices in the hybrid EA training process suggests that this algorithm may not be trapped in the false minima of the error surface [15]. The performance of hybrid evolutionary algorithm has been found better in comparison to the genetic algorithm by finding the less number of iterations and higher rate of convergence for character recognition, on comparing the performance of two different networks (first with five neurons in two hidden layers and the second with six neurons in two hidden layers) it has been found that the network with six neurons in two hidden layers provide the better results. So it is expected that the network consisting with six neurons in two hidden layers will provide the optimum network [15].

In this chapter the trained neural network for straight hand written alphabets are used for training and recognition of scaled and / or rotated alphabets. The scaled and/ or rotated samples used for training have been taken from the trained set of straight alphabets while the test patterns has been considered from the set of test pattern of straight alphabets and again these patterns have been scaled and rotated to consider them as the new training pattern set for refinement . Then, the new samples of each alphabet have been used as test patterns for this problem. The training is done by genetic algorithm and hybrid evolutionary algorithm for five different trials of trained network. It has been observed that the performance of online training by hybrid evolutionary algorithm [16] is found better by getting less number of iterations and more number of convergence weight matrices in comparison to random genetic algorithm. The rate of recognition of scaled and / or rotated hand written English alphabets is also found very high.

Second section of this chapter deals with the basic mathematics of image scaling and rotations. Section three defines the present status of the trained network, while the fourth section defines the design of network and implementation details. Section five shows the results of the experiments. In section six discussions have been made and the section seven shows the future works.

5.2 Scaling and Rotation

The alphabets used as pattern, first scanned and then converted to bitmap image so the patterns of handwritten English alphabets will be consider as the images.

In geometry and linear algebra, the scaling is defined [17, 18] as mathematics to resize the image onto the X and Y axis (for two dimensional objects). The more obvious and common way to change the size of an image is scaling the image. The content of the image is enlarged by increasing the size of the pixel values of the image. But while the actual image pixels and colors are modified, the content represented by the image is essentially left unchanged. Scaling of an image make a drastic change to the content of the image. An image can be represented as pixel matrix and a scaling can be represented by a scaling matrix. While the rotation [17, 19] is a transformation in a plane that describes the change in the orientation of an image. A rotation is different from a translation, which has no fixed points, and from a reflection, which "flips" the image it is transforming. This overall process of change the shape, position and orientation of an image is known as the transformation of image. When an image is moved to stationary co-ordinate system or background, referred as geometric transformation and applied to each point of an image. And while the co-ordinate system is moved relative to the image and image is held stationary then this process is termed as a co-ordinate transformation. In the process of transformation every image is assumed as a set of points. Every point P of the image has co-ordinate (x, y) and the complete image is defined as sum of total of all co-ordinates points [20].

Image rotation is a transformation where the image is rotated θ^o about origin [21]. The co-ordinate of new point is given by the following equation, $P' = R_\theta(P)$

Let us consider that the co-ordinate of a point P is (x, y) and after rotation the coordinate of point P' is (x', y'). If initially P is at the α angle from x-axis then

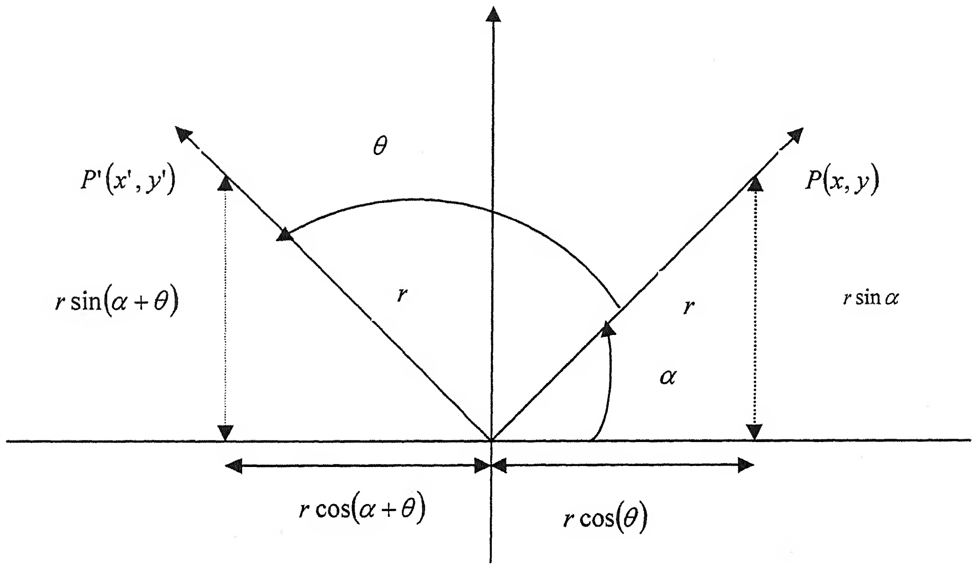


Figure 5.1: The diagrammatical representation of rotation.

$$x' = r\cos(\alpha+\theta)$$
$$= r\cos\alpha\cos\theta - r\sin\alpha\sin\theta$$
$$\sin ce \quad x = r\cos\alpha \quad and \quad y = r\sin\alpha$$
$$therefore \quad x' = x\cos\theta - y\sin\theta \tag{5.1}$$
$$and \quad y' = r\sin\alpha\cos\theta + r\sin\theta\cos\alpha$$
$$y' = x\sin\theta + y\cos\theta$$

So, the rotation matrix can be defined as $R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, by multiplying this to the original image we can get a rotated image.

While the scaling is the process of changing the size and proportion of the image [18]. There are two factors used in scaling transformation i.e. S_x and S_y, where S_x is a scale factor for the x co-ordinate and S_y is scale factor of y co-ordinate.Let us assume that the co-ordinate of a point P is (x, y) and after scaling the coordinate of point P' is (x', y') then,

$$P' = S_{S_x,S_y}(P) \tag{5.2}$$

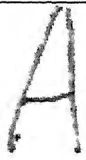
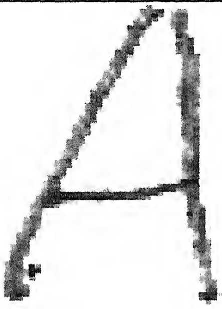
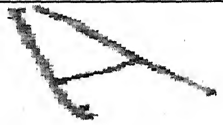
where

$$x' = S_x.x \quad and$$
$$y' = S_y.y \tag{5.3}$$

or we may define the scaling matrix as $S_{S_x,S_y} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$, so by multiplying the original image with the scaling matrix we can get scaled image.

As already mentioned that the alphabets first scanned then converted to the bitmap image, so every alphabet is defined as an image. For the scaling and rotation, the above mentioned mathematics have been applied by MATLAB software. The nature and the density values of alphabets changed after rotation and / or scaling. It can easily be shown as:

Table 5.1: The comparison table to show the change in the nature of character.

	Alphabets	Density values Matrix	
Straight Sample		2.466523 2.350251	2.190908 2.361397
Scaled 2X3 Sample		2.409672 2.455222	2.550000 2.550000
Rotate By 30°		2.302031 2.262767	2.443418 2.399731

5.3 Present Status of the Trained Network

We have already discussed [14] the problem of straight hand written character recognition. The result for training pattern set and the test pattern set have been analyzed with the three different algorithm(BP, Random GA, Hybrid EA).In this chapter we consider the two neural network architecture. The training set of the hand written English alphabets have been considered as the straight alphabets i.e. without scaling and rotation. There are two network have been trained with these set of training set. As each network consist with two hidden layers with input and output layers. The difference between these two architectures used in terms of number of neurons in each hidden layer the first architecture has used five neurons in each hidden layer while the second architecture used six neurons in each hidden layer. The analysis of the training and testing for these two architecture with the three algorithms(BP, Random GA, and Hybrid EA),suggested that the network with six neurons in each hidden layer performs better in terms of number of iterations and number of converged weight vectors. The performance of the network is found better with the hybrid evolutionary algorithm in comparison to other backpropagation (BP) and random genetic algorithm (GA). So that, on the basis of this analysis, we consider the neural network with six neurons in two hidden layers (as shown in figure 5.2), for the training of the new pattern set.

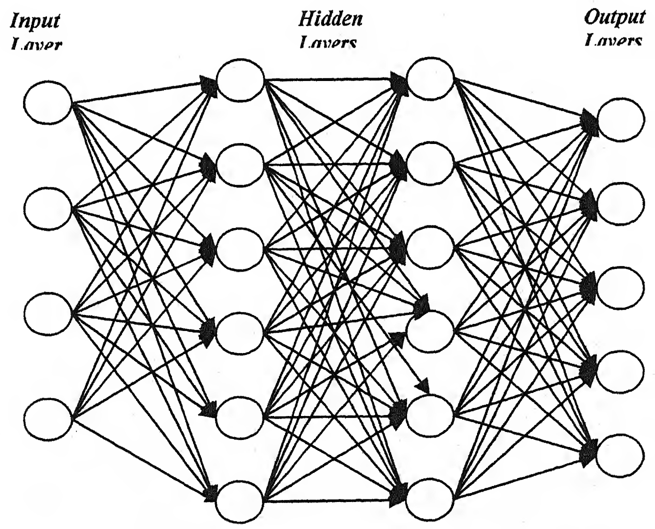


Figure 5.2: The Architecture of Neural Networks used.

The selected pattern set consists with the handwritten English alphabets with scaling and rotation. The patterns those have been already used for training previously now scaled and rotated to construct the new training pattern set. We have analyzed[14] that during the recognition of straight hand written English alphabets (A to Z), on five trials for each training set the total number 1475(approx.) of converged weights obtained for hybrid evolutionary algorithm and 965 (approx.) for random genetic algorithms. Thus there are 1475 and 965 number of optimal weight vectors explored, on which (for any one of them) all the patterns of training set have properly trained and also the performance has verified from the test pattern set. Hence from the above mentioned analysis, it is quite clear that any one of the weight vector (Converged or Optimal) can be selected for further training for the training set consists with the scaled and rotated form of the already used training set. The chromosome of the selected weight vector is shown in figure 5.3,

W_{11}^{IH}	.....	W_{41}^{IH}	***Bias of Hidden unit 1 of layer 1***	W_{21}^{IH}	......	W_{42}^{IH}	***Bias of Hidden unit 2 of layer 1***	....	W_{16}^{IH}	......	W_{46}^{IH}	***Bias of Hidden unit 6 of layer 1***

W_{11}^{HO}	......	W_{61}^{H0}	***Bias of unit 1 of Output layer***	W_{12}^{HO}	......	W_{62}^{H0}	***Bias of unit 1 of Output layer***	....	W_{15}^{HO}	....	W_{65}^{HO}	***Bias of unit 5 of output layer***

W_{11}^{HH}	......	W_{61}^{HH}	***Bias of Hidden unit 1 of layer 2***	W_{21}^{HH}		W_{62}^{HH}	***Bias of Hidden unit 2 of layer 1***	....	W_{16}^{HH}	......	W_{66}^{HH}	***Bias of Hidden unit 6 of layer 2***

Figure 5.3: The Chromosome of selected weight vector.

The training set is considering form the already trained pattern from existing training set of hand written English alphabets. Thus any one of the already trained pattern for each alphabet (A to Z) has considered an It has been scaled and rotated. This new pattern has been used for further training with two algorithms (Random GA and Hybrid EA) & performance of the network has been analyzed as shown in table (alphabets table).

Accomplish the training the following operators and parameters shown in table 5.2 and 5.3 have been considered.

Table 5.2: Genetic operators used in the experiments

Training Algorithms	**Genetic Operators Used**
GA	Mutation with probability <= 0.1 and Crossover
Hybrid EA	Mutation with probability <= 0.1 and Crossover

Table 5.3: Parameters used for genetic algorithm and hybrid evolutionary algorithm

Parameter	**Value**
Adaptation Rate (K)	3.0
Learning Rate (η)	0.1
Momentum Descent Term (α)	0.9
Doug's Momentum Term $\left(\frac{1}{1-(\alpha)}\right)$	$\left(\frac{1}{1-(\alpha)}\right)$

Mutation Population Size	3
Crossover Population Size	2000
Minimum Error Exist in the Network $(MAXE)$	0.00001
Initial weights and biased term values	Randomly Generated Values Between 0 and 1

5.4 Simulation Design and Implementation Details

In this simulation design, as we mentioned already that the network with four neurons in the input layer, two hidden layers with six neurons in each and five neurons in the output layer, will be considered for the training purpose with the two algorithms i.e. random genetic algorithm and hybrid evolutionary algorithm. The input / output pattern vectors are constructed from the already existing training pattern set of straight alphabets. The network has already been trained for the existing training set for the five trials of each pattern. Now we select any one of the pattern vector for each alphabet and apply the scaling & rotation randomly on the selected pattern. This modified pattern now becomes the new pattern for training set. The network is trained for this new training set with already converged weight vectors, in this manner the network is continuously adapting the change in its behavior corresponding to changing training set [13] .Thus the training pattern set is continuously increasing with the newly modified pattern. The network is also expected to adopt the continuous changes of the training pattern set. Again, as the training process will progress the more and more new converged weight vectors will explore. The performance of the network will analyzed with respect to random GA and hybrid EA in terms of the number of iterations and converged weight vectors. Thus, after completion of the training up to a satisfactory level, a new randomly generated pattern (Not used in the training set) has used as test pattern .The performance of the network can analyze with respect to both algorithms for the straight and rotated/scaled test patterns

5.4.1 Experiment

In the previous paper five different sets of alphabets were used for the two networks with three different algorithms i.e. BP, random GA, and hybrid EA. The alphabets were converted into their density function by using MATLAB program, for input data. In the each experiment we had taken five trials for the training of the neural network population. One of the five samples of each alphabet has been picked up randomly then scaling and rotation functions have been applied through MATLAB program. The scaled and rotated characters have been converted to their density function by using the same MATLAB program which was used earlier to find the density values of straight alphabets, these density values then used as a new input training pattern. While for the new test pattern set, the new alphabets (not form the trained set of straight alphabets) have been picked up and the same process is repeated as stated above to generate the new input test patterns.

All the input sample of alphabets to convert their density values were partitioned into four equal parts and the density values of the pixels for each part were calculated. Next, the density values of the central of gravities for these partitioned samples were calculated. Consequently four values were obtained from a sample, which were then used as the input patterns for the feed-forward neural network. This procedure was used to present the input pattern to the feed-forward neural network for each of the samples.

5.4.2 The Neural Network Architecture

The architecture of the neural networks was based on a fully connected feed-forward multilayer generalized perceptron [13]. The hidden layers were used to investigate the effects of the algorithms on the hyper plane. Each network had a single output unit with the following activation and output functions,

$$A_k^o = \sum_{i=0}^{H} w_{io_k} O_i^h \tag{5.3}$$

$$\int\left(A_k^o\right) = \int\left(\sum_{i=0}^{H} w_{io_k} O_i^h\right) = O_k^o \tag{5.4}$$

where function $\int\left(A_k^o\right)$ is given as,

$$O_k^o = \frac{1}{1+e^{-KA_k^o}} \tag{5.5}$$

Now, similarly, the output and activation value for the neurons of hidden layers and input layer can be written as,

$$A_k^h = \sum_{i=0}^{N} w_{ik} O_i \tag{5.6}$$

and $$O_k^h = \frac{1}{1+e^{-KA_k^h}} \tag{5.7}$$

and $$O_k^i = \int\left(A_k^i\right) = A_k^i \tag{5.8}$$

For the online training [13], the change in weight populations was done according to the calculated error in the network after each of the iteration of training. The change in weight and error in the network can be calculated as follows,

$$\Delta w_{ho}(s+1) = -\eta \sum_{i=1}^{H} \frac{\partial E}{\partial w_{ho}} + \alpha \Delta w_{ho}(s) + \frac{1}{1-(\alpha \Delta w_{ho}(s))} \quad (5.9)$$

$$\Delta w_{ih}(s+1) = -\eta \sum_{i=1}^{N} \frac{\partial E}{\partial w_{ih}} + \alpha \Delta w_{ih}(s) + \frac{1}{1-(\alpha \Delta w_{ho}(s))} \quad (5.10)$$

$$E = \frac{1}{2} \sum_{p=1}^{P} \left(O^{o} - T\right)^{2} \quad (5.11)$$

where $\left(O^{o} - T\right)^{2}$ is the squared difference between the actual output value of the output layer for pattern p and the target output value. Doug's momentum term was used with momentum descent term for calculating the change in weights in equations (5.9) and (5.10). Doug's momentum descent is similar to standard momentum descent with the exception that the pre-momentum weight step vector is bounded so that its length cannot exceed 1.0. After the momentum is added, the length of the resulting weight change vector can grow as high as 1 / (1 - momentum). This change allows stable behavior with much higher initial learning rates, resulting in less need to adjust the learning rate as the training progresses. The evolutionary algorithms evolve the population of weights using its operators, and select the best population of the weights that minimize the error between the desired output and the actual output of neural network system.

5.4.3 The Genetic Algorithm Implementation

A mutation operator which randomly selects a gene in a chromosome and adds a small random value between –1 and 1 to that particular gene produces the next generation population of 107-gene chromosomes. The size of the next generated population will be $n+1$, if the mutation operator applied n times over the old chromosome:

$$C^{new} = C^{old} \bigcup_{i=1}^{n} \left[C_{121-\lambda} \bigcup \left(C_{\lambda}{}^{old} + \in \right) \right] \tag{5.12}$$

where C^{old} symbolizes the old chromosome of 107-gene, $\in$ symbolizes the small randomly generated value between –1 to 1, λ symbolizes the randomly selected gene of C^{old} chromosome for adding the $\in$, and C^{new} symbolizes the next generation population of chromosome, i.e. $C^{new} = [C_1, C_2, C_3, -------, C_n, C_{n+1}]$. The inner $\bigcup$ operator prepares a new chromosome at each iteration of mutation, and the outer $\bigcup$ operator builds the new population of chromosomes called C^{new}.

Elitism was used when creating each generation so that the genetic operators did not lose good solutions. This involved copying the best-encoded network unchanged into the new population as given in equation. 3.10, which includes C^{old} for creating C^{new}.

Selection of a chromosome C^{sel} is made from the mutated population of chromosomes for which the sum of squared errors is minimum for the feed-forward neural network, i.e. iteratively all the chromosome values will be assigned to the network architecture in terms of weights and biased values defined in chromosome. After assigning the values, the network architecture will be able to fabricate output using these assigned values. For each new chromosome C^{new}, the error can be calculated using these fabricated outputs as in equation 2.7. Next, the selection operator will pick a chromosome C^{sel} from C^{new}, which generates minimized error for the network.

Crossover operator takes selected chromosome and creates a child for producing the next generation population of 107-gene chromosomes of size $n+1$. This next generation of population is produced by applying the crossover operator n times. Experiments performed the crossover operation by swapping the two randomly selected gene values of the parent chromosome as given in Fig. 3.4 and equation 3.11:

$$C^{next} = C^{sel} \bigcup_{i=1}^{n} \left[\left(C^{sel} - C_{\alpha}^{sel} - C_{\beta}^{sel} \right) \cup \left(C_{\alpha}^{sel} \xleftrightarrow[v_{\alpha} \leftrightarrow v_{\beta}]{} C_{\beta}^{sel} \right) \right] \tag{5.13}$$

where α and β symbolize the randomly generated genes positions in CP_1 and CP_2 in C^{sel} chromosome, and C^{next} is the next generation of size $n+1$.

Chromosome C^{sel}

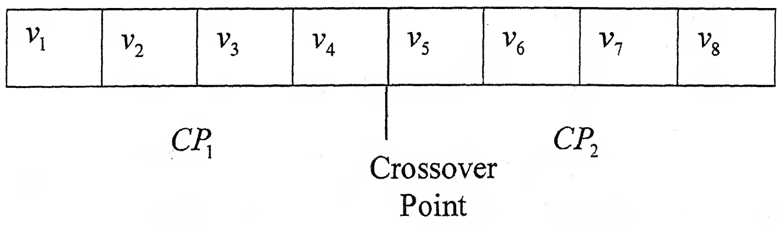


Figure 5.4(A)

Chromosome

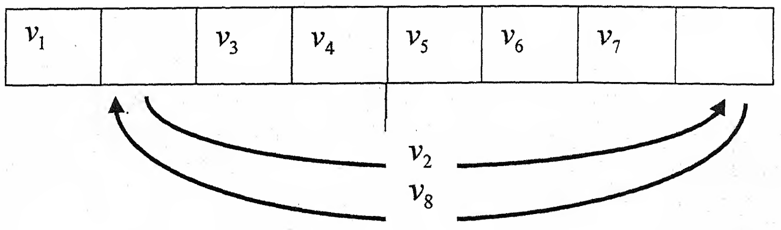


Figure 5.4(B)

Chromosome	v_1	v_8	v_3	v_4	v_5	v_6	v_7	v_2

Figure 5.4(C)

Figure 5.4: (A) Chromosome before applying crossover operator, (B) Applying crossover operator on chromosome, and (C) Chromosome after applying crossover operator

A fitness evaluation function defines a function for evaluating the chromosome performance. This function must estimate the performance of weight population of a given feed-forward neural network. A simple function defined based on the proportion of the sum of squared errors is applied. To evaluate the fitness of a given chromosome, each weight and biased value contained in the chromosome is assigned to the respective link and neuron in the network. The training set is then presented to the network, and the sum of squared errors is calculated. The smaller the sum, the higher is the fitness of the chromosome. In other words, the genetic algorithm attempts to find a set of weights and biased values that minimizes the sum of squared errors.

$$\min error = 1.0$$

For all the n+1 chromosomes

$$\text{if } \left(\min error > E_{C_i^{next}}\right) \text{ then } \quad \begin{array}{l} \left(\min error = E_{C_i^{next}}\right) \\ C^{\min} = C_i^{next} \end{array}$$

$$\text{else} \quad \left(\min error = \min error\right) \tag{5.14}$$

where $E_{C_i^{next}}$ symbolizes the error calculated for i^{th} chromosome among the $n+1$ chromosomes of C^{next} population and $C^{\min}$ symbolizes the chromosome which has minimized error.

5.5 Results and Discussions

The results of the experiment are shown below consist of eight tables [5.5 to 5.12] and ten figures [5.5 to 5.20]. Tables [5.5 & 5.6] contain the entries for average number of iterations performed by the hybrid evolutionary algorithm and random genetic algorithm respectively for the training pattern set of straight alphabets and scaled & rotated alphabets. Tables [5.7 & 5.8] contain the entries for average number of convergence weight matrices obtained by the hybrid evolutionary algorithm and random genetic algorithm respectively for the training pattern set of straight alphabets and scaled & rotated alphabets. Tables [5.9 & 5.10] contain the entries for average number of iterations performed by the hybrid evolutionary algorithm and random genetic algorithm respectively for the test pattern set of straight alphabets and scaled & rotated alphabets. Tables [5.11 & 5.12] contain the entries for average number of convergence weight matrices obtained by the hybrid evolutionary algorithm and random genetic algorithm respectively for the test pattern set of straight alphabets and scaled & rotated alphabets. While the figures [5.5 to 5.12] show the comparison charts between the performance of training and recognition of straight and scaled & rotated pattern based on the tables [5.5-5.12]. While the figures [5.13 to 5.16] show the comparison chart between the performance of the algorithms for trained pattern set and test pattern set., and the Figures [5.17 to 5.20] show the comparison chart between the performance of algorithms used.

Table 5.5: The iteration performed for training set of straight and scaled and rotated alphabets for hybrid evolutionary algorithm.

Sample (Straight Alphabets)	Average Iterations for Five Trails of Five Samples of Each Alphabets	Sample	Scaling	Rotation	Average Iterations for five trials
A	27	A1	2 by 3	0	15
B	16	B2	2 by 2	50	78
C	88	C3	3 by 3	35	5
D	32	D4	1 by 2	22	356
E	20	E2	2 by 2	65	1003
F	14	F2	2 by 1	45	252
G	12	G1	3 by 3	20	81
H	14	H4	1 by 1	35	16
I	31	I1	3 by 2	25	26
J	77	J2	2 by 3	30	119
K	56	K3	2 by 3	20	4
L	20	L4	2 by 2	35	80
M	30	M2	3 by 3	25	18
N	63	N2	2 by 3	20	69
O	4	O1	2 by 2	25	14
P	12	P4	2 by 1	65	30

Q	10	Q1	3 by 3	15	181
R	4	R2	2 by 2	25	57
S	22	S3	1 by 2	25	220
T	17	T4	2 by 3	30	52
U	38	U2	2 by 3	20	79
V	23	V5	2 by 2	35	196
W	18	W1	3 by 3	25	4
X	11	X4	2 by2	45	27
Y	19	Y5	3 by 3	15	6
Z	9	Z3	2 by 3	30	158

Figure 5.5: The comparison chart between the average numbers of iterations performed to recognize the straight and scaled & rotated alphabets by hybrid evolutionary algorithm.

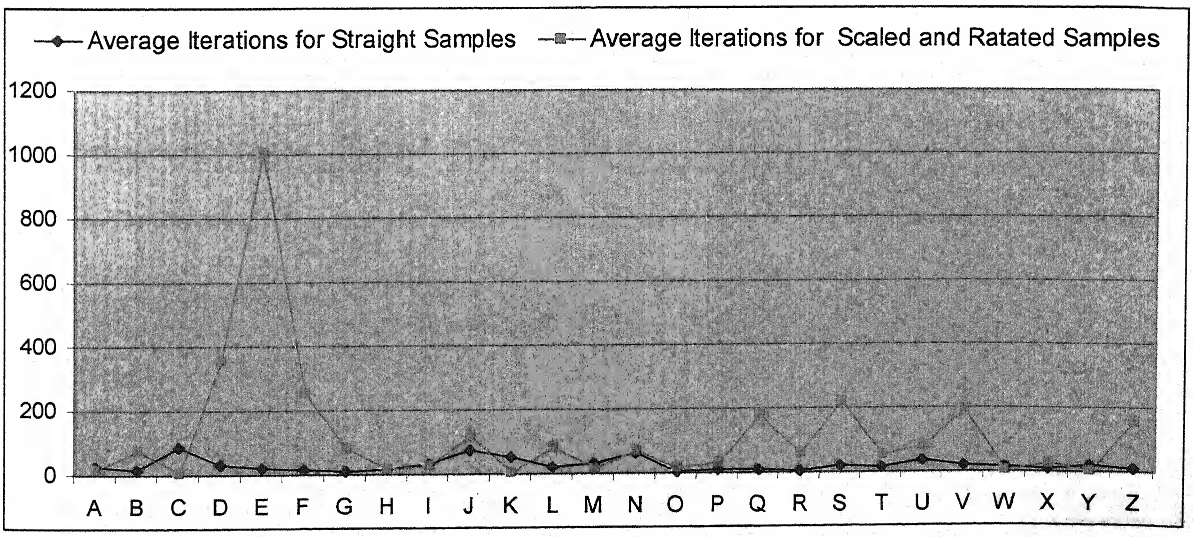


Table 5.6: The iteration performed for training set of straight and scaled and rotated alphabets for random genetic algorithm.

Sample (Straight Alphabets)	Average Iterations for Five Trails of Five Samples of Each Alphabets	Sample	Scaling	Rotation	Average Iterations for five trials
A	1552	A1	2 by 3	0	814
B	610	B2	2 by 2	50	159
C	678	C3	3 by 3	35	1132
D	400	D4	1 by 2	22	1077
E	1161	E2	2 by 2	65	364
F	569	F2	2 by 1	45	126
G	67	G1	3 by 3	20	663
H	2482	H4	1 by 1	35	271
I	384	I1	3 by 2	25	108
J	1365	J2	2 by 3	30	684
K	4053	K3	2 by 3	20	426
L	408	L4	2 by 2	35	765
M	2171	M2	3 by 3	25	574
N	53	N2	2 by 3	20	76
O	19	O1	2 by 2	25	81
P	81	P4	2 by 1	65	391

Q	523	Q1	3 by 3	15	1185
R	397	R2	2 by 2	25	1506
S	108	S3	1 by 2	25	524
T	6636	T4	2 by 3	30	122
U	2612	U2	2 by 3	20	411
V	7048	V5	2 by 2	35	2399
W	70	W1	3 by 3	25	81
X	9317	X4	2 by2	45	975
Y	8385	Y5	3 by 3	15	461
Z	5298	Z3	2 by 3	30	699

Figure 5.6: The comparison chart between the average numbers of iterations performed to recognize the straight and scaled & rotated alphabets by hybrid evolutionary algorithm.

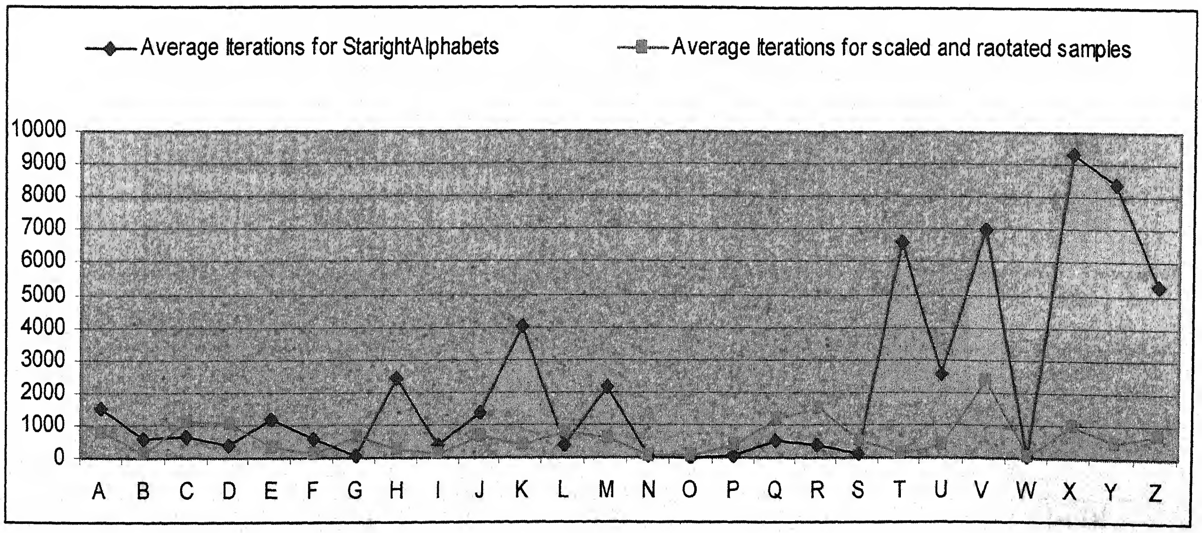


Table 5.7: The number convergence vectors obtained for training set of straight and scaled and rotated alphabets for hybrid evolutionary algorithm.

Sample (Straight Alphabets)	Average number of convergence weight Vectors for Five Trails of Five Samples of Each Alphabets	Sample	Scaling	Rotation	Average convergence weight Vectors for five trials
A	1262	A1	2 by 3	0	368
B	1479	B2	2 by 2	50	749
C	1548	C3	3 by 3	35	373
D	1734	D4	1 by 2	22	1840
E	1474	E2	2 by 2	65	1778
F	1496	F2	2 by 1	45	1112
G	1434	G1	3 by 3	20	746
H	1737	H4	1 by 1	35	1113
I	1752	I1	3 by 2	25	1126
J	1538	J2	2 by 3	30	1091
K	1555	K3	2 by 3	20	381
L	1403	L4	2 by 2	35	1375
M	1260	M2	3 by 3	25	363
N	1337	N2	2 by 3	20	1091
O	1394	O1	2 by 2	25	385
P	1756	P4	2 by 1	65	1448

Q	1411	Q1	3 by 3	15	1107
R	1292	R2	2 by 2	25	1082
S	1601	S3	1 by 2	25	1479
T	1216	T4	2 by 3	30	730
U	1284	U2	2 by 3	20	1445
V	1648	V5	2 by 2	35	53.2
W	1410	W1	3 by 3	25	409
X	1358	X4	2 by2	45	1391
Y	1297	Y5	3 by 3	15	1055
Z	1692	Z3	2 by 3	30	665

Figure 5.7: The comparison chart between the average numbers of convergence weight matrix obtained to recognize the straight and scaled & rotated alphabets by hybrid evolutionary algorithm.

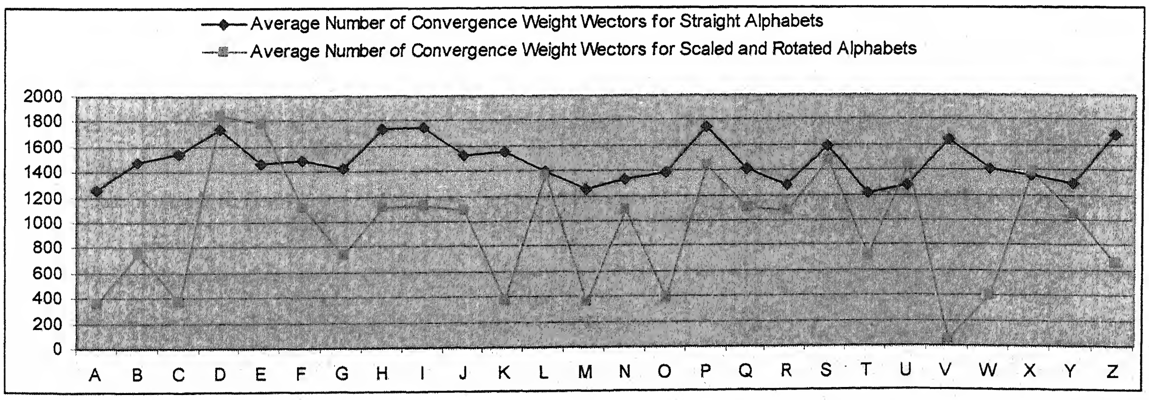


Table 5.8: The number convergence vectors obtained for training set of straight and scaled and rotated alphabets for random genetic algorithm.

Sample (Straight Alphabets)	Average number of convergence weight vectors for Five Trails of Five Samples of Each Alphabets	Sample	Scaling	Rotation	Average number of convergence weight vectors for five trials
A	848	A1	2 by 3	0	1027
B	1037	B2	2 by 2	50	521
C	916	C3	3 by 3	35	940
D	981	D4	1 by 2	22	854
E	1193	E2	2 by 2	65	776
F	849	F2	2 by 1	45	794
G	910	G1	3 by 3	20	936
H	811	H4	1 by 1	35	546
I	1046	I1	3 by 2	25	580
J	981	J2	2 by 3	30	732
K	1209	K3	2 by 3	20	1143
L	1047	L4	2 by 2	35	544
M	850	M2	3 by 3	25	960
N	646	N2	2 by 3	20	575
O	929	O1	2 by 2	25	866
P	512	P4	2 by 1	65	762

Q	953	Q1	3 by 3	15	864
R	1116	R2	2 by 2	25	589
S	1186	S3	1 by 2	25	1055
T	1067	T4	2 by 3	30	280
U	890	U2	2 by 3	20	559
V	1137	V5	2 by 2	35	910
W	898	W1	3 by 3	25	655
X	1018	X4	2 by2	45	1013
Y	997	Y5	3 by 3	15	278
Z	1068	Z3	2 by 3	30	593

Figure 5.8: The comparison chart between the average numbers of convergence weight matrix obtained to recognize the straight and scaled & rotated alphabets by genetic algorithm.

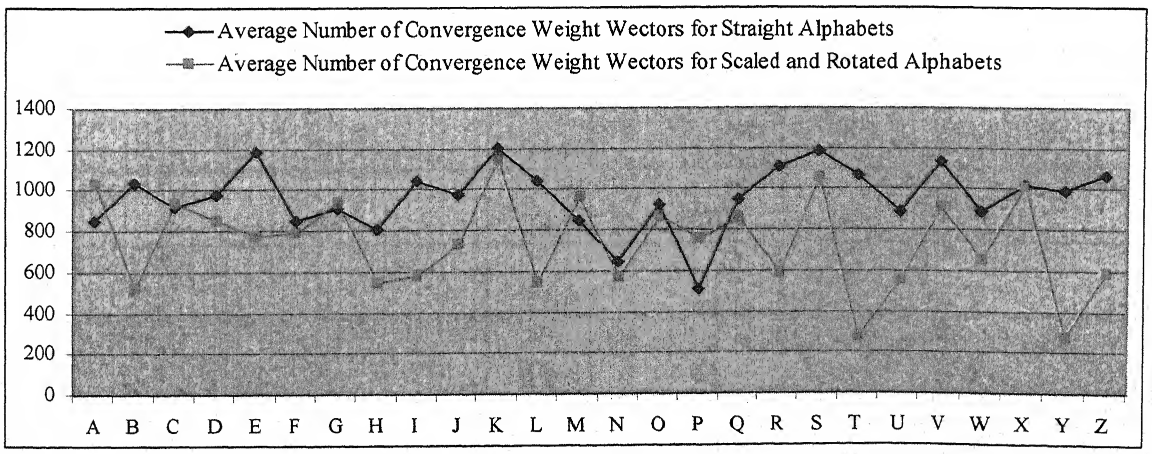


Table 5.9: The average iterations for five trials, performed for test pattern set of straight and scaled & rotated alphabets for hybrid evolutionary algorithm.

Alphabets	Average Iterations of Straight Test Alphabets	Scaling	Rotation	Average Iterations after Scaling and Rotation
A	73	2 by 1	45	4
B	1	2 by 3	25	2
C	2	2 by 2	36	9
D	4	3 by 3	45	9
E	2	3 by 3	55	6
F	4	2 by 2	35	1
G	395	2 by 2	30	2
H	7	2 by 3	35	37
I	1	2 by 2	45	1
J	1	2 by 2	35	1
K	24	2 by 2	30	1
L	2	2 by 3	15	3
M	23	2 by 2	20	1
N	2	3 by 3	25	1
O	2	3 by 3	45	1
P	21	2 by 2	35	2
Q	1	2 by 2	20	2

R	1	3 by 2	65	21
S	1	2 by 2	45	51
T	45	2 by 2	35	19
U	76	2 by 1	25	6
V	212	2 by 3	15	1
W	1	2 by 2	20	1
X	3	2 by 3	55	1
Y	222	3 by 2	30	1
Z	3	2 by 2	45	1

Figure 5.9: The comparison chart between the iterations performed to recognize the test pattern, of straight and scaled & rotated alphabets by hybrid evolutionary algorithm.

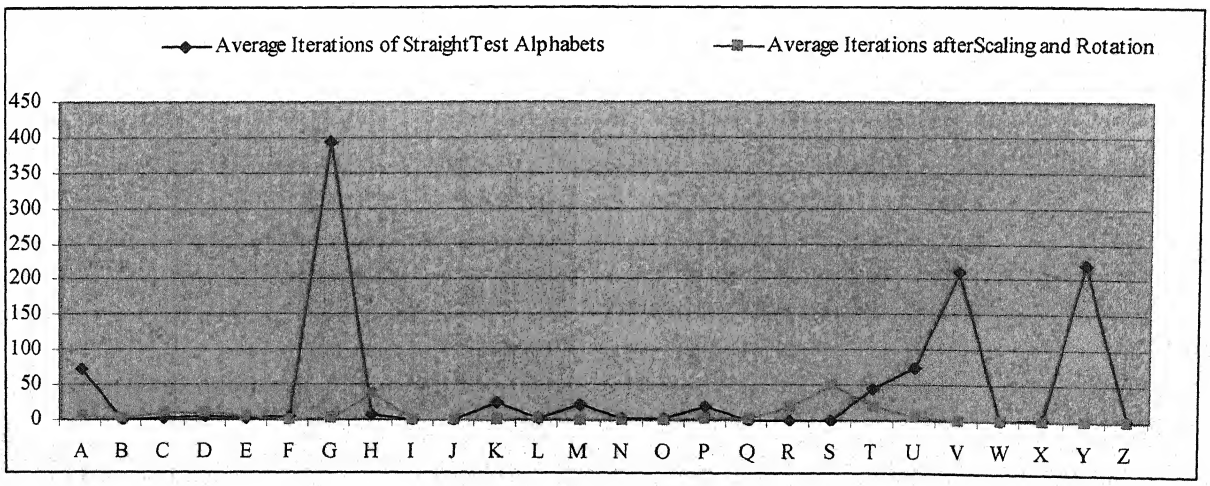


Table 5.10: The average iterations for five trials performed for test pattern set of straight and scaled & rotated alphabets for genetic algorithm.

Alphabets	Average Iterations of Straight Test Alphabets	Scaling	Rotation	Average Iterations after Scaling and Rotation
A	68	2 by 1	45	320
B	47	2 by 3	25	43
C	3207	2 by 2	36	831
D	189	3 by 3	45	4
E	240	3 by 3	55	4
F	95	2 by 2	35	382
G	113	2 by 2	30	1
H	72	2 by 3	35	15
I	2309	2 by 2	45	68
J	45	2 by 2	35	1
K	9	2 by 2	30	45
L	50	2 by 3	15	58
M	120	2 by 2	20	12
N	32	3 by 3	25	24
O	24	3 by 3	45	143
P	34	2 by 2	35	1
Q	22	2 by 2	20	2

R	588	3 by 2	65	232
S	29	2 by 2	45	89
T	114	2 by 2	35	1
U	42	2 by 1	25	215
V	23	2 by 3	15	1
W	3	2 by 2	20	33
X	159	2 by 3	55	64
Y	180	3 by 2	30	13
Z	387	2 by 2	45	283

Figure 5.10: The comparison chart between the iterations performed to recognize the test pattern, straight and scaled & rotated alphabets by random genetic algorithm.

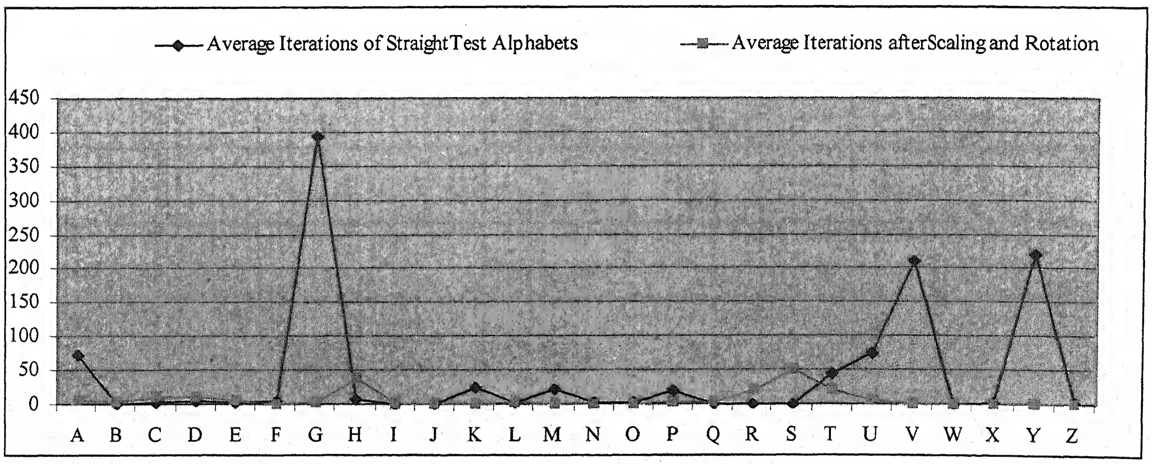


Table 5.11: The average number of convergence weight matrix for five trials, obtained for test pattern set of straight and scaled & rotated alphabets for hybrid evolutionary algorithm.

Alphabets	Average Number of Convergence Weight Matrix Straight Test Alphabets	Scaling	Rotation	Average Number of Convergence Weight Matrix after Scaling and Rotation
A	1121	2 by 1	45	4
B	1	2 by 3	25	2
C	27	2 by 2	36	9
D	2	3 by 3	45	9
E	57	3 by 3	55	6
F	10	2 by 2	35	1
G	1705	2 by 2	30	2
H	1	2 by 3	35	37
I	1	2 by 2	45	1
J	14	2 by 2	35	1
K	1904	2 by 2	30	1
L	1831	2 by 3	15	3
M	1849	2 by 2	20	1
N	8	3 by 3	25	1
O	1794	3 by 3	45	1

P	1828	2 by 2	35	2
Q	11	2 by 2	20	2
R	4	3 by 2	65	21
S	1790	2 by 2	45	51
T	1783	2 by 2	35	19
U	1748	2 by 1	25	6
V	1790	2 by 3	15	1
W	26	2 by 2	20	1
X	9	2 by 3	55	1
Y	1874	3 by 2	30	1
Z	1	2 by 2	45	1

Figure 5.11: The comparison chart between the iterations performed to recognize the test pattern, straight and scaled & rotated alphabets by hybrid evolutionary algorithm.

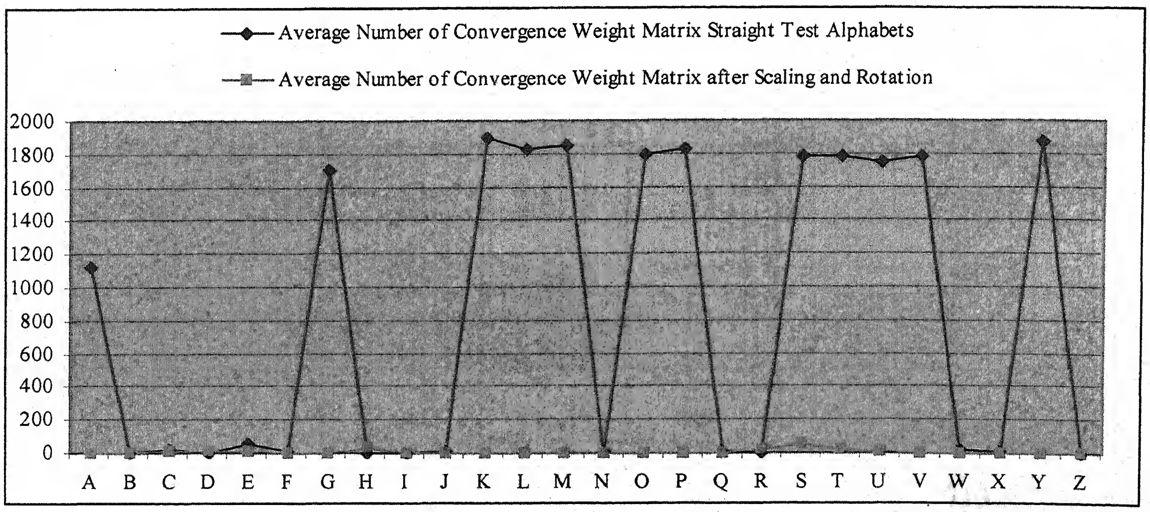


Table 5.12: The average number of convergence weight matrix for five trials, obtained for test pattern set of straight and scaled & rotated alphabets for genetic algorithm.

Alphabets	Average Number of Convergence Weight Matrix Straight Test Alphabets	Scaling	Rotation	Average Number of Convergence Weight Matrix after Scaling and Rotation
A	2	2 by 1	45	1003
B	3	2 by 3	25	858
C	1	2 by 2	36	848
D	4	3 by 3	45	804
E	5	3 by 3	55	1029
F	1	2 by 2	35	1027
G	1	2 by 2	30	604
H	10	2 by 3	35	596
I	905	2 by 2	45	769
J	1	2 by 2	35	692
K	1	2 by 2	30	874
L	1	2 by 3	15	1031
M	1	2 by 2	20	683
N	6	3 by 3	25	745
O	2	3 by 3	45	644
P	2	2 by 2	35	713

Q	9	2 by 2	20	1163
R	1	3 by 2	65	873
S	5	2 by 2	45	1373
T	5	2 by 2	35	270
U	3	2 by 1	25	664
V	13	2 by 3	15	924
W	1	2 by 2	20	1168
X	1	2 by 3	55	1400
Y	11	3 by 2	30	1178
Z	1	2 by 2	45	1182

Figure 5.12: The comparison chart between the iterations performed to recognize the test pattern, straight and scaled & rotated alphabets by random genetic algorithm.

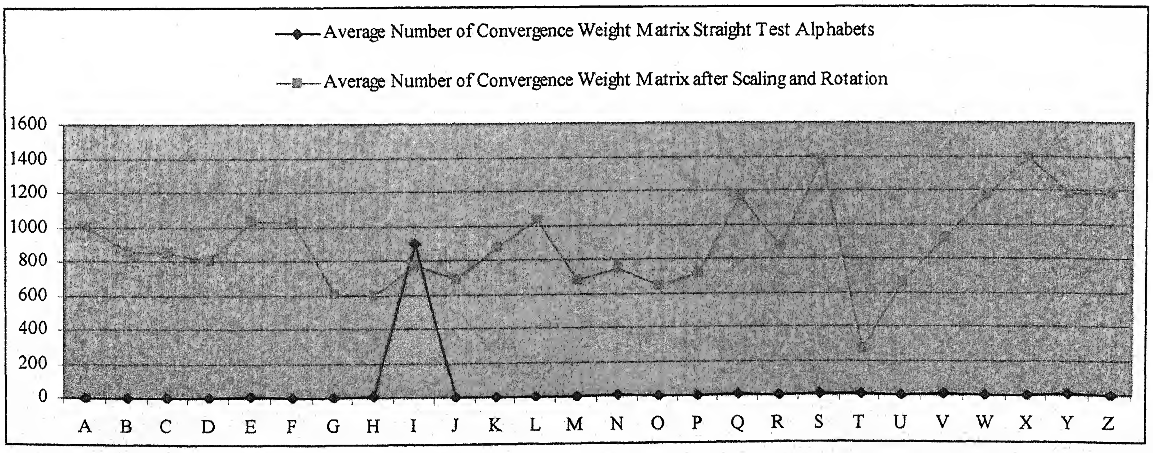


Figure 5.13: The comparison chart between the iterations performed to recognize the training set pattern with test pattern set for scaled & rotated alphabets by hybrid evolutionary algorithm.

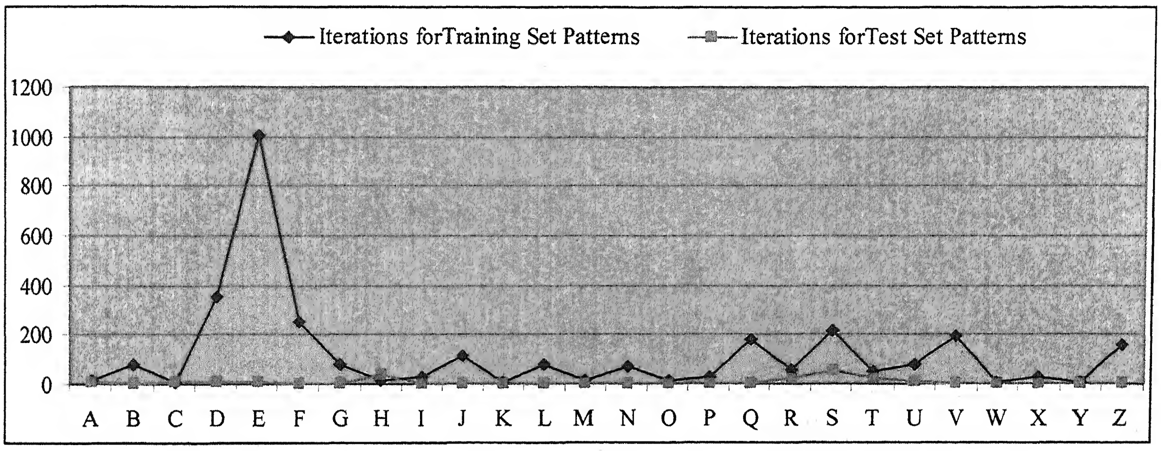


Figure 5.14: The comparison chart between the iterations performed to recognize the training set pattern with test pattern set for scaled & rotated alphabets by genetic algorithm.

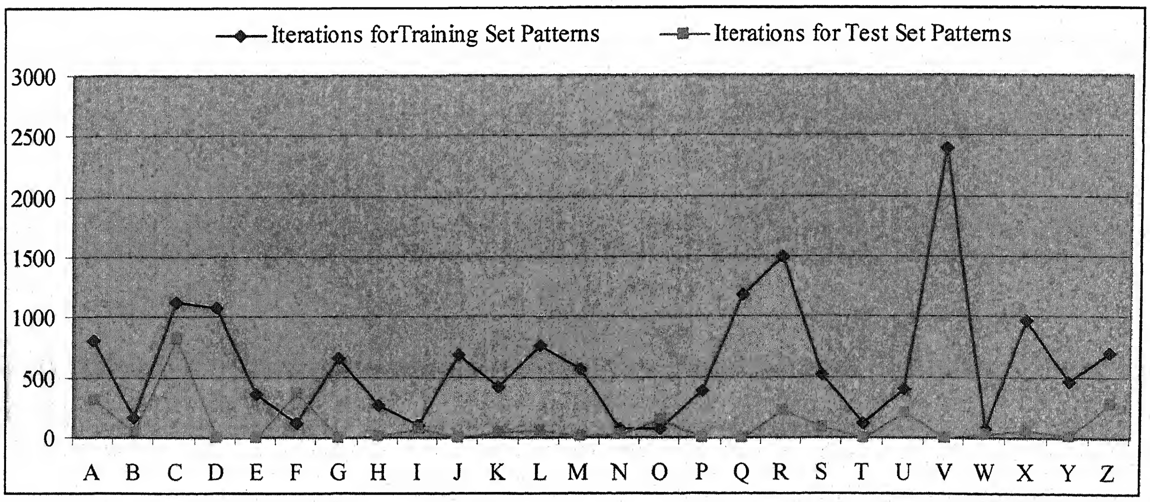


Figure 5.15: The comparison chart between the numbers of convergence weight matrices obtained to recognize the training set pattern with test pattern set for scaled & rotated alphabets by hybrid evolutionary algorithm.

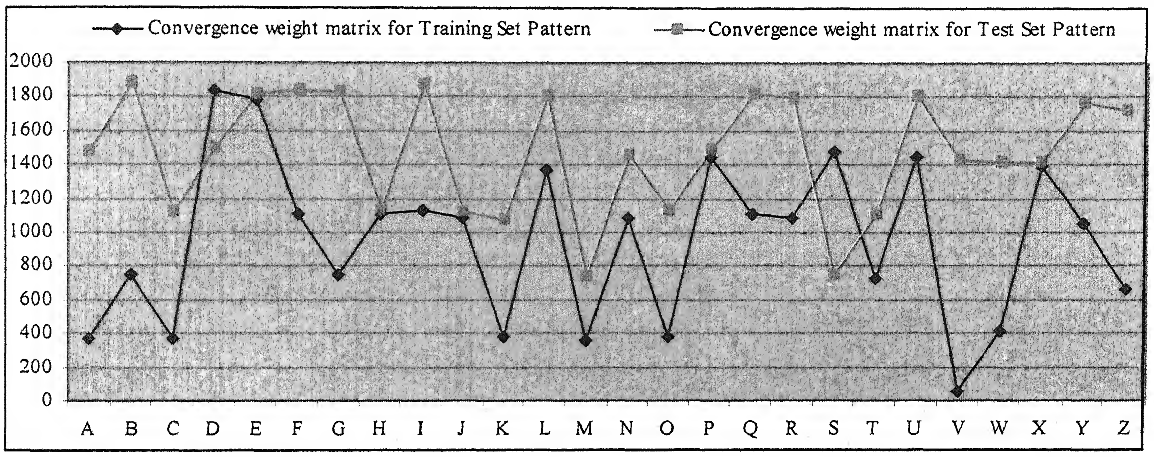


Figure 5.16: The comparison chart between the numbers of convergence weight matrices obtained to recognize the training set pattern with test pattern set for scaled & rotated alphabets by genetic algorithm.

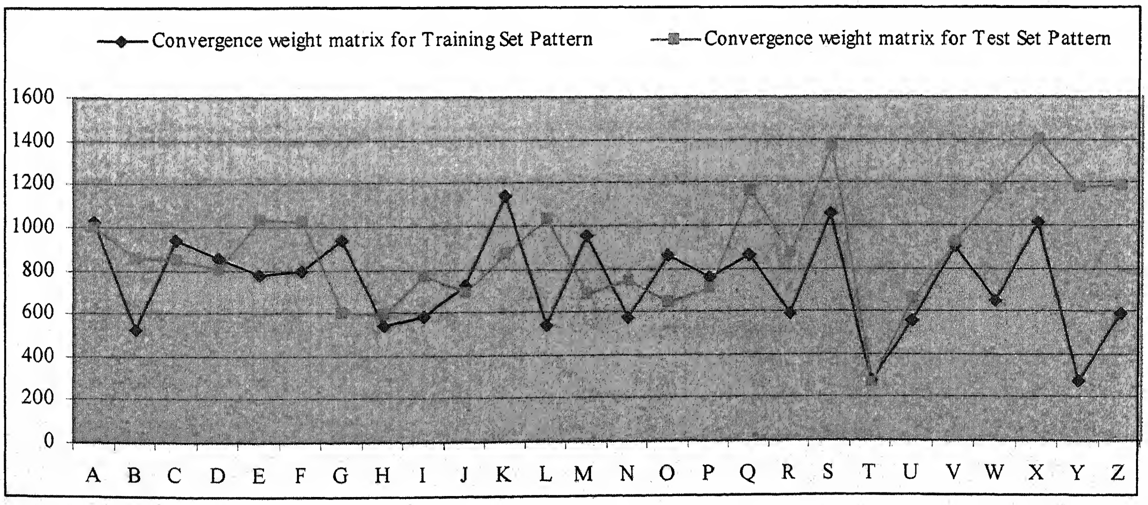


Figure 5.17: The comparison chart between the numbers of iterations performed by random genetic algorithm and by hybrid evolutionary algorithm to recognize the training set pattern for scaled & rotated alphabets.

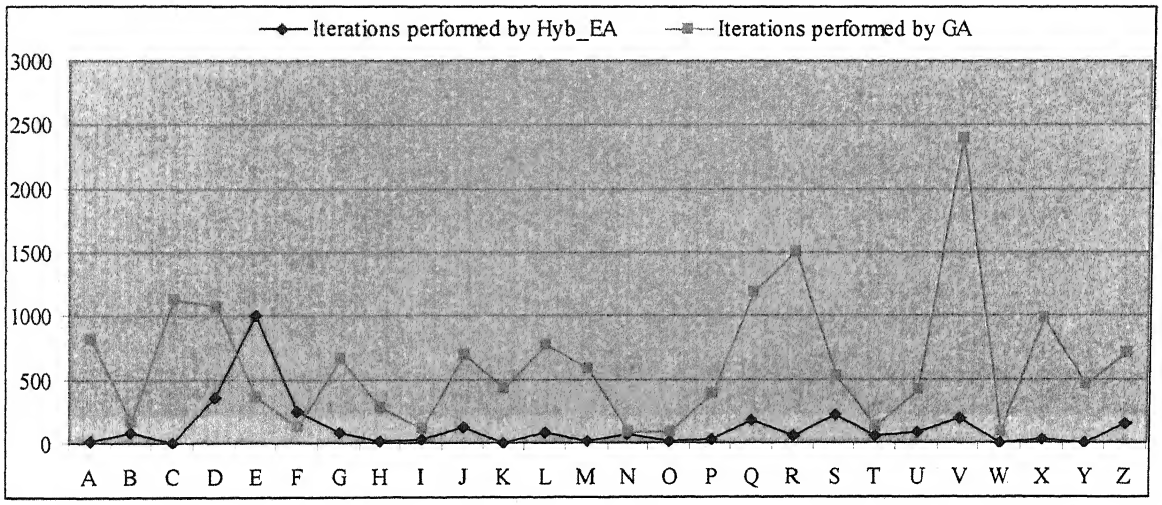


Figure 5.18: The comparison chart between the numbers of iterations performed by random genetic algorithm and by hybrid evolutionary algorithm to recognize the test set pattern for scaled & rotated alphabets.

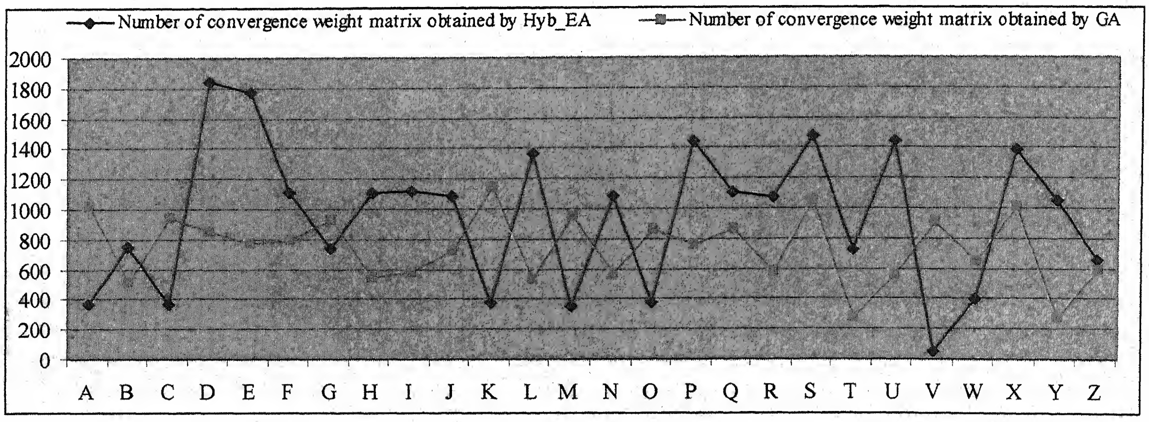


Figure 5.19: The comparison chart between the numbers of convergence weight matrices obtained by random genetic algorithm and by hybrid evolutionary algorithm to recognize the training set pattern for scaled & rotated alphabets.

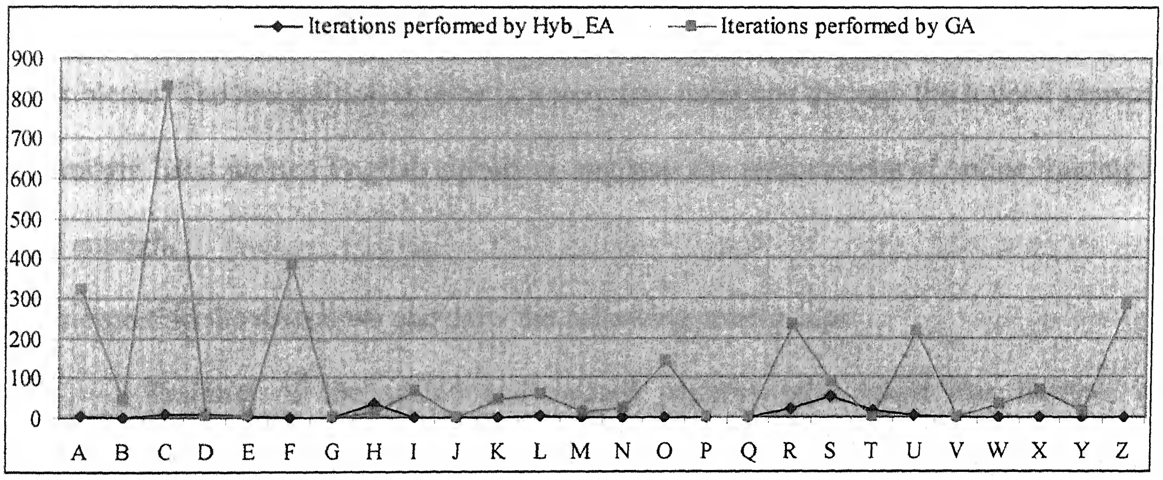
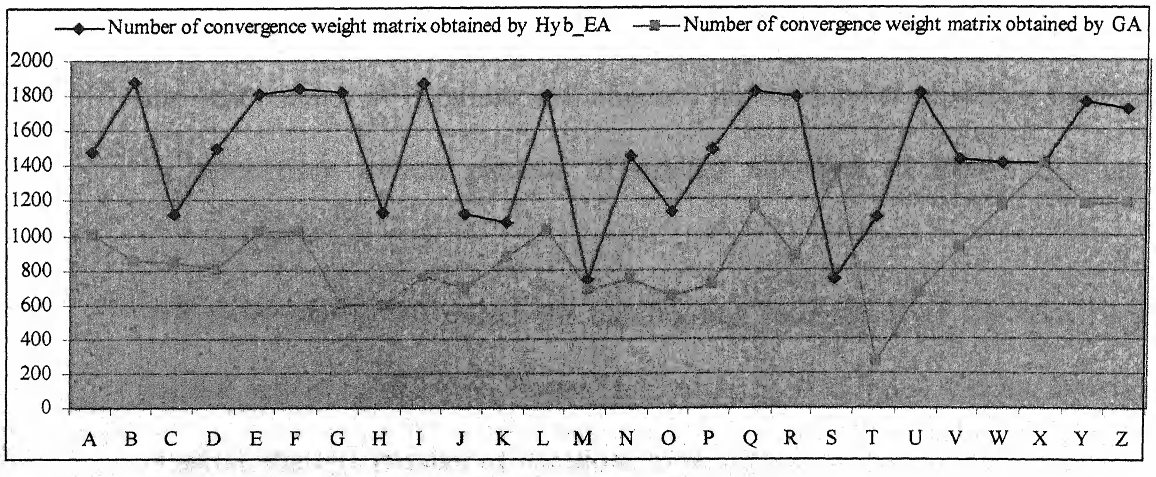


Figure 5.20: The comparison chart between the numbers of convergence weight matrices obtained by random genetic algorithm and by hybrid evolutionary algorithm to recognize the test set pattern for scaled & rotated alphabets.



5.6 Conclusion

The results and the comparison charts mentioned above clearly show that the on-line training to recognize the scaled and rotated hand written English alphabets by genetic algorithm and hybrid evolutionary algorithm, provide a better scope to solve these kind of problems. The recognition is done in a very few iterations through the trained network for straight hand written English alphabets, suggests the performance of online training is good enough.

By interpreting the results we can draw the following conclusions:

1. Training of the scaled and rotated patterns have taken less number of iterations and obtained higher number of weight matrices in comparison to straight patterns as shown in Figure [5.5 to 5.8] for both hybrid evolutionary algorithm and genetic algorithm. It shows that the training of the network is accurate and the network can easily adapt the new similar pattern.
2. The recognition of test pattern of scaled and rotated alphabets have also taken the less number of iterations and obtained higher number of weight matrices in comparison to test patterns of straight alphabets as shown in figure [5.9 to 5.12].
3. While comparing the results for training and recognition of the scaled and rotated pattern as shown in Figure [5.13 to 5.16], it is found that the network is taking the less number of iterations to recognize the pattern in comparison to the number of iterations performed for the training of the network and also

obtains the higher number of convergence weight matrices. It also leads the same conclusion that the training of the network is accurate and reliable.

4. By comparing the results of the hybrid evolutionary algorithm and the genetic algorithm Figure [5.17 to 5.20], it is found that the performance of hybrid evolutionary algorithm is far better in comparison to the genetic algorithm.

5.7 Future Work

The successful implementation of feed forward neural networks with evolutionary algorithms for hand written English (straight and scaled and /or rotated) alphabets, motivated to apply the same for other complex problems of pattern recognition. It is expected to extend the work in future for the following problems:

1. Recognition of overlapped alphabets.
2. Refinement of training of the incomplete training set due to wage ness, fuzzy ness, or some other complexity in the samples.

References:

[1] Impedovo, S. (editor), "*Fundamentals in Handwriting Recognition.*" ,NATO-Advanced Study Institute Series F. Springer-Verlag, (1994).

[2] Mori, S., Suen, C. Y., and Yamamoto, K., "Historical review of OCR research and development". *Proceedings of the IEEE (*Special Issue on Optical Character Recognition.*)*, vol. 80(7), pp. 1029--1058, (1992).

[3] Fukushima, K and Wake, N., "Handwritten alphanumeric character recognition by the neocognitron", IEEE Trans. on Neural Networks, vol. 2(3), pp. 355-365, (1991).

[4] Blackwell, K. T., Vogl, T. P. and Hyman, S. D., Barbour, G. S., and Allcon, D. L., "A new approach to handwritten character recognition", Pattern Recognition, vol. 25(6), pp. 655-666, (1992).

[5] Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbad, W. and Jackel, L. D., "Handwritten digit recognition with a backpropagation network", Neural Information Processing Systems, Touretzky editor, vol. 2, pp. 396-404, Morgan Kaufmann Publishers, (1990).

[6] Ahmed, M. and Ward, R., "An Expert System for General Symbol Recognition.", Pattern Recognition, vol. 33(12), pp. 1975-1988, (2000).

[7] Amin, A., "Recognition of Hand-Printed Latin Characters Based on Generalized Hough Transform and Decision Tree Learning Techniques.", International Journal of Pattern Recognition and Artificial Intelligence, vol. 14(3), pp. 369-387, (2000).

[8] Hanmandlu, M. ,Murali Mohan, K.R. and Kumar, H., "Neural based handwritten character recognition", Proceedings of the Fifth International Conference on Document Analysis and Recognition(ICDAR), pp. 241 – 244, (1999)

[9] Hailong, Liu and Xiaoqing, D.," Handwritten character recognition using gradient feature and quadratic classifier with multiple discrimination schemes", Proceedings, Eighth International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp.: 19 - 23 (2005).

[10] Srikantan, G., Lam, S. W., and Srihari, S.N.,"Gradient-based contour encoding for character recognition.",Pattern Recognition, vol. 29,pp. 1147–1160,(1996).

[11] Kharma, N., and Ward, R. , "A Novel Invariant Mapping Applied to Handwritten Arabic Character Recognition", Pattern Recognition, vol. 34(11), August, (2001).

[12] Badi, K. and Shimura, M.,"Machine Recognition of Arabic Cursive Scripts.", In Transactions of the Institute of Electronics & Communications Engrs. Japan, vol. E65, pp. 107-114, (1982).

[13] Yegnarayana, B.,"Artificial Neural Networks", Prentice Hall of India,(2004).

[14] Shrivastava, S. and Singh, M.P" Analysis of pattern classification for the hand written English alphabets with soft computing approach.", Communicated, Applied Soft Computing Journals of Elseviers Publication, July,(2007).

[15] Shrivastava, S. and Singh, M.P., "Analysis of Soft Computing Techniques for Minmizing the Problem of Local Minima in Back Propagation for Hand written

English Alphabets.", Proceedings, International Conference on Soft Computing and Intelligent Systems, vol. 2, pp. 307-313,(2007)

[16] Montana,D.J., and Davis, L., "Training feedforward networks using genetic algorithms.", In: Sridhara N (ed), Proceedings 11th International Joint Conf. on Artificial Intelligence, pp. 762–767, (1989).

[17] "Transformation matrix" - Wikipedia, the free encyclopedia.htm

[18] "Scaling (geometry)" - Wikipedia, the free encyclopedia.htm

[19] "Rotation (mathematics)" - Wikipedia, the free encyclopedia.htm

[20] Annadurai, S. and Shanmugalakshmi, R.,"Fundamental of Digital Image Processing.", Pearson Education, ISBN 81-7758-479-0,(2007)

[21] Vince, J.A., "Mathematics for Computer Graphics.", Springer, ISBN 10:1-84-628-034-6,(2006)

APPENDIX

List of Papers Published and Communicated in the Proceedings of International Conferences and International Journals.

1. Shrivastava, S. and Singh, M.P" Analysis of pattern classification for the hand written English alphabets with soft computing approach.", ***Communicated***, Applied Soft Computing Journals of Elseviers Publication, July,(2007).
2. Shrivastava, S. and Singh, M.P., "Analysis of Soft Computing Techniques for Minmizing the Problem of Local Minima in Back Propagation for Hand written English Alphabets.", Proceedings, International Conference on Soft Computing and Intelligent Systems, vol. 1, pp. 307-313,(2007).
3. Shrivastava, S. and Singh M.P., "Analysis for the Recognition of Scaled and Rotated Hand Writtn English Alphabets with Online Learning Implementation using Soft Computing Techniques.", ***Communicated***, International Journal of Artificial Intelligence and Tools (IJAIT), Word Scientific, (2008).